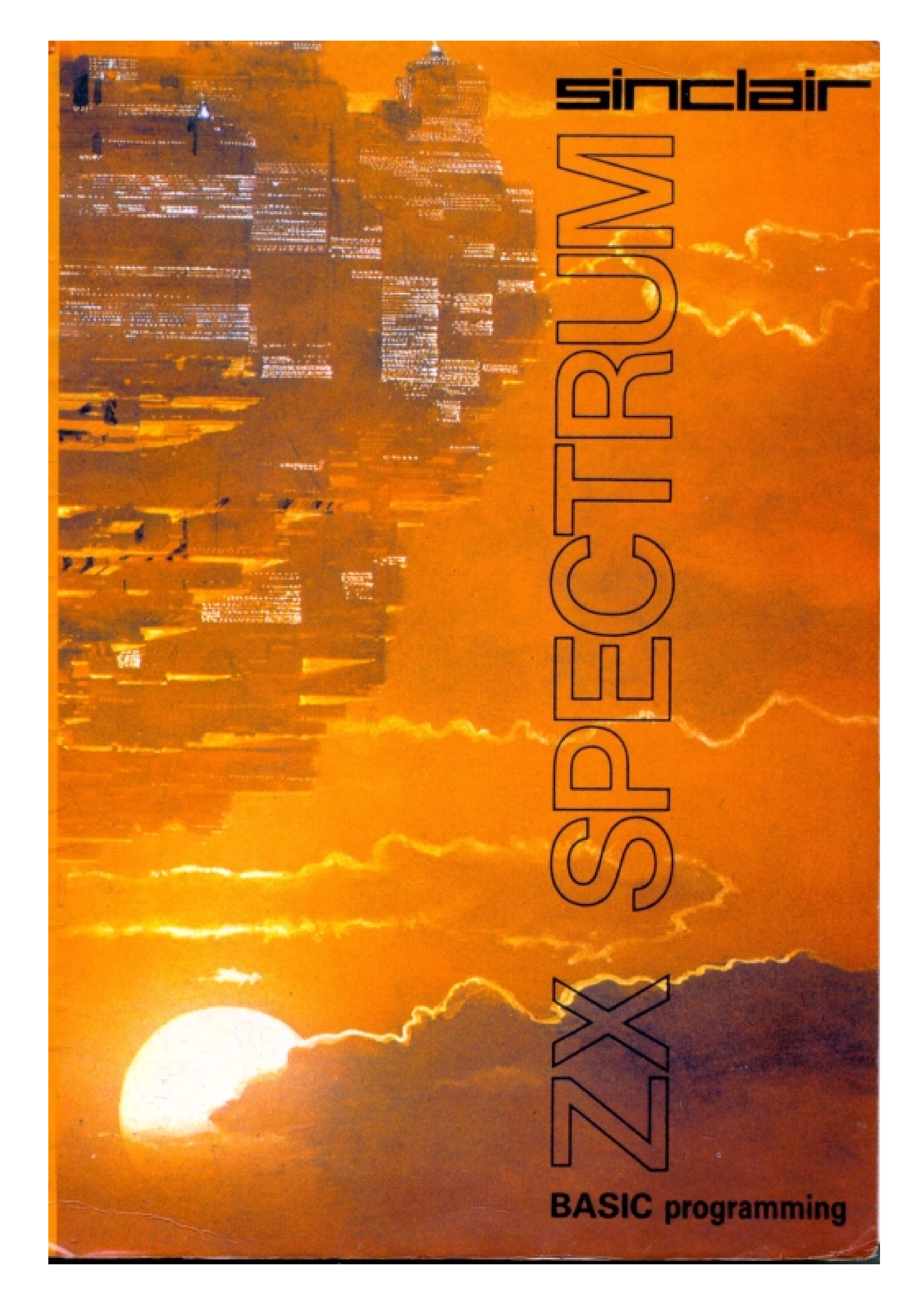
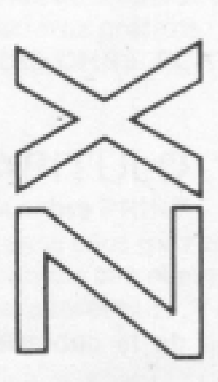

sinclair
ZX SPECTRUM
BASIC programming

sinclair

ZX SPECTRUM



INTRODUCTION

por Steven Vickers
y Robin Bradbeer

TRADUCIDO Y ADAPTADO POR

INVESTRONICA, S. A.

PRIMERA EDICION 1982
SEGUNDA EDICION 1983

Depósito Legal: M. 32.590-1983
Imprime: Mateu Cromo, S. A. Pinto (Madrid)



Ilustración de la cubierta por John Harris de Young Artists.

Indice

CAPITULO

1

Introducción

Tanto si leyó primero el fascículo de introducción como si pasó directamente a este manual, debe tener conocimiento de que los comandos se obedecen inmediatamente y que las instrucciones comienzan con un número de línea y se almacenan para su posterior ejecución. Asimismo, debe conocer los comandos **PRINT, LET e INPUT** que pueden utilizarse en todas las máquinas que usan BASIC) y **BORDER, PAPER** y **BEEP** (que se utilizan en el Spectrum).

Este manual de BASIC comienza repitiendo algunas cosas dadas en el fascículo de introducción, pero con mucho más detalle, indicándole con exactitud lo que usted puede hacer y lo que no puede. También encontrará algunos ejercicios al final de cada capítulo. No haga caso omiso de ellos, pues muchos de ellos ilustran puntos que están solamente insinuados en el texto. Écheles una ojeada y haga alguno que le interese o que le parezca referirse a un tema que no entiende adecuadamente.

Siempre que pueda hacerlo y, en cualquier caso, tenga encendido el ordenador. Si se pregunta: ¿Qué hará el ordenador si le digo esto o lo otro?, no vacile en introducirlo por el teclado y ver lo que ocurre. Siempre que el manual le diga que teclee algo, pregúntese a sí mismo: ¿Qué podría teclear en su lugar? y pruebe sus respuestas. Cuantos más programas propios escriba, tanto mejor comprenderá el funcionamiento del ordenador.

Al final de este manual de programación hay algunos apéndices. En ellos se incluyen secciones sobre la forma en que está organizada la memoria, sobre cómo el ordenador manipula los números y una serie de programas ejemplo que ilustran la potencia del ZX Spectrum.

El teclado

Los caracteres del ZX Spectrum no solamente comprenden los símbolos simples (letras, dígitos, etc.), sino también los comandos (palabras clave, nombres de funciones, etc.), estos últimos accesibles con una sola pulsación de la tecla en lugar de deletrearse. Para obtener todas estas funciones o los comandos, algunas teclas tienen cinco o más significados distintos, dados en parte por el cambio de las teclas (esto es, pulsando la tecla **CAPS SHIFT** o la tecla **SYMBOL SHIFT** al mismo tiempo que la requerida) y en parte, al tener la máquina en modos operativos diferentes.

El modo viene indicado por el cursor, que es una letra parpadeante que indica en dónde se insertará el siguiente carácter introducido por el teclado.

El modo K (por «Keywords» - Palabras clave) sustituye automáticamente al modo L cuando la máquina está esperando un comando o una línea de programa (en vez de entrada de datos), y por su posición en la línea sabe que debe esperar un número de línea o una palabra clave. Esta posición es al principio de la línea o inmediatamente después de **THEN** o de **:** (salvo en una cadena). Si no está cambiada la siguiente tecla se interpretará como una palabra clave (escrita en las teclas) o bien como un dígito.

El modo L (por letras) suele presentarse en todas las demás circunstancias. Si no está cambiada la siguiente tecla se interpretará como el símbolo principal en esa tecla, en minúsculas para las letras.

En los modos K y L, **SYMBOL SHIFT** y una tecla serán interpretados como el carácter rojo subsidiario en la tecla y **CAPS SHIFT** con una tecla de dígito se interpretará como la función de control escrita en blanco en la tecla. **CAPS SHIFT** con las demás teclas no afecta a las palabras clave en el modo K y, en el modo L, convierte las minúsculas en mayúsculas.

El modo C (por «Capitals» - letras mayúsculas) es una variante del modo L, en donde todas las letras aparecen como mayúsculas. **CAPS LOCK** produce un cambio del modo L al modo C o viceversa.

El modo E (por extendido) se utiliza para obtener caracteres adicionales, la mayoría comandos. Tiene lugar después de que se pulsen juntas ambas teclas de cambio y dura solamente el intervalo de una pulsación de tecla. En este modo, una letra proporciona un solo carácter o comando (que se indica en verde encima de ella) si no está cambiada, y otro (que se indica en rojo debajo de ella), si se pulsa con un cambio. Una tecla de dígito proporciona un comando, si se pulsa con **SYMBOL SHIFT;** de cualquier modo, da una secuencia de control de colores.

El modo G (por gráficos) se produce después de la pulsación de **GRAPHICS (CAPS SHIFT** y **9)** y dura hasta que se vuelva a pulsar otra vez o se pulse 9 en sí mismo. Una tecla de dígito proporcionará un gráfico de mosaico, liberada la tecla **GRAPHICS** o **DELETE,** y cada una de las teclas de letra, aparte de V, W, X, Y y Z, dará un gráfico definido por el usuario.

Si cualquier tecla se mantiene oprimida durante más de 2 ó 3 segundos, comenzará la repetición correspondiente.

La introducción por el teclado aparece en la mitad inferior de la pantalla a medida que se teclea, insertándose cada carácter (símbolo simple o comando) inmediatamente antes del cursor. El cursor puede desplazarse a la izquierda con **CAPS SHIFT** y **5,** o a la derecha con **CAPS SHIFT** y **8.** El carácter antes del cursor puede borrarse con **DELETE (CAPS SHIFT** y **0**). Nota: La línea completa puede borrarse tecleando **EDIT (CAPS SHIFT** y **1)** y a continuación, **ENTER.**

Cuando se pulsa **ENTER,** se ejecuta la línea, se introduce en el programa o se utiliza como entrada de datos, cuando sea adecuado, a no ser que contenga un error de sintaxis. En este caso, aparece un signo **?** parpadeante junto al error.

A medida que se introducen las líneas del programa, se visualiza un listado en la mitad superior de la pantalla. La última línea introducida se denomina **línea en curso** y se indica por el símbolo **>** ; este puede desplazarse con el empleo de las teclas **↓ (CAPS SHIFT** y **6)** y **↑ (CAPS SHIFT** y **7).** Si se pulsa **EDIT (CAPS SHIFT** y **1**), la línea en curso se duplica en la parte inferior de la pantalla y puede corregirse.

Cuando se ejecuta un comando o un programa, el resultado se visualiza en la mitad superior de la pantalla y se mantiene allí hasta que se introduzca una línea de programa, o se pulse **ENTER** con una línea vacía o se pulse **↑** ó **↓.** En la parte inferior aparece un informe que da un código (dígito o letra) al que se hace referencia en el Apéndice B. El informe permanece en la pantalla hasta que se pulse una tecla (e indique el modo K).

En determinadas circunstancias, **CAPS SHIFT** con la tecla **SPACE** actúa como **BREAK,** deteniendo el ordenador con informe D o L. Ello se identifica

(i) al final de una sentencia mientras se está ejecutando un programa o
(ii) mientras el ordenador esté utilizando la impresora o grabadora de cassette.

La pantalla de televisión

Tiene 24 líneas, con 32 caracteres cada una de longitud, y está dividida en dos partes. La parte superior comprende, como máximo, 22 líneas y visualiza un listado o salida de programa. Cuando la impresión en la parte superior ha llegado a la parte inferior, todo se desplaza hacia arriba en una línea; si ello trajera consigo la pérdida de una línea que no hubiera tenido todavía oportunidad de ver, entonces el ordenador se detiene con el mensaje **scroll?**. Al pulsar las teclas **N, SPACE** o **STOP** se hará que se pare el

programa con el informe **D BREAK - CONT repeats;** cualquier otra tecla permitirá que continúe el desplazamiento hacia arriba («scrolling»). La parte inferior se emplea para la introducción de comandos, lineas de programa o entrada de datos y también para visualizar informes: La parte inferior se inicia a partir de dos líneas (la superior es una línea en blanco), pero se amplía para admitir lo que pueda introducirse por el teclado. Cuando llegue a la posición de la impresión en curso en la mitad superior, ulteriores ampliaciones harán que se desplace hacia arriba la mitad superior.

CAPITULO

2

Conceptos de la programación en Basic

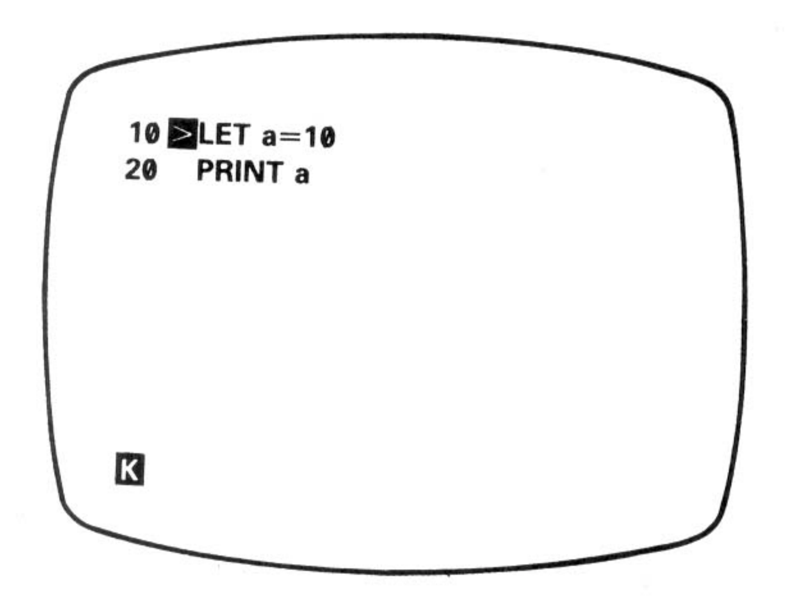
Resumen

Programas
Números de línea
Edición correctora de programas con el
empleo de ↓ , ↑ y **EDIT, RUN, LIST**
GO TO, CONTINUE, INPUT, NEW, REM, PRINT
STOP en entrada de datos
BREAK

Teclee estas dos líneas de un programa de ordenador para obtener el resultado de la suma de dos números:

20 PRINT a
10 LET a = 10

de modo que en la pantalla aparezca:



Como ya sabe, debido a que estas líneas comenzaban con números, no se «obedecieron» inmediatamente sino que se almacenaron como líneas de programa. También habrá observado que los números de línea rigen el orden de las instrucciones del. programa, lo cual es muy importante a la hora de ejecutarlo, pero que también se refleja en el orden de las líneas en el listado que puede ver ahora en la pantalla.

Hasta ahora sólo ha introducido un número, por lo que teclee:

15 LET b = 15

y veamos lo que ocurre. Hubiera sido imposible insertar esta línea entre las dos antes indicadas si estas se hubieran numerado 1 y 2, en lugar de 10 y de 20 (los números de línea deben ser números enteros entre 1 y 9999), por ello es por lo que, cuando se introduce por el teclado un programa, es muy conveniente dejar espacios entre los números de línea.

Ahora necesita cambiar la línea 20 a:

20 PRINT a + b

Podría introducir la nueva instrucción por completo, pero resulta más fácil emplear el modo **EDIT** descrito en el fascículo de introducción. El > en la línea 15 se denomina el cursor del programa y la línea a la que apunta es la línea en curso. Esta suele ser la última línea que introdujo, pero puede utilizar las teclas ↑ o ↓ para desplazar el cursor del programa hacia abajo o hacia arriba (inténtelo, dejando el cursor del programa eventualmente en la línea 20).

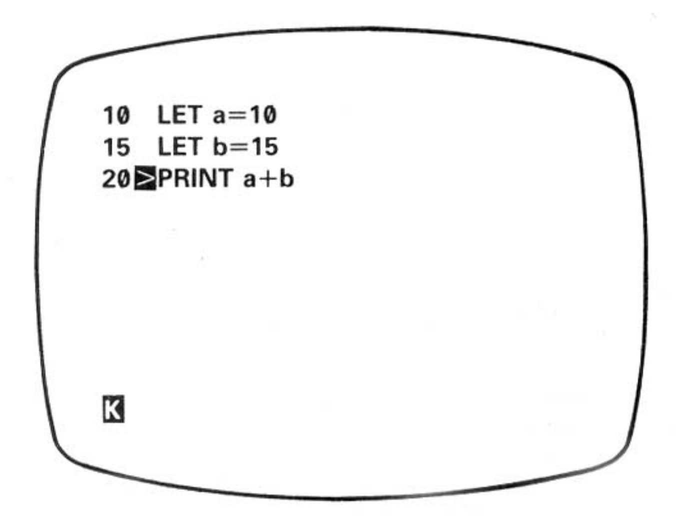
Cuando pulse la tecla **EDIT**, una copia de la línea en curso se visualizará en la parte inferior de la pantalla (en su caso, una copia de la línea 20). Mantenga oprimida la tecla → hasta que el cursor L se desplace al final de la línea y entonces, teclee

+b (sin **ENTER)**

La línea en la parte inferior de la pantalla debe leerse ahora:

20 PRINT a + b

Pulse **ENTER** y sustituirá la anterior línea 20, de modo que en la pantalla aparecerá:



Ejecute este programa con el empleo de **RUN** y **ENTER** y se visualizará la suma. Vuelva a ejecutar el programa y luego teclee:

PRINT a,b

Las variables siguen estando allí, aun cuando el programa haya acabado.

Hay un método útil empleando **EDIT** para prescindir de la parte inferior de la pantalla. Teclee cualquier cosa que carezca de significado (sin **ENTER**), e imagínese que después desea hacerlo desaparecer por carecer de valor. Una forma de borrarlo es pulsar la tecla **DELETE,** pero de otro modo de hacerlo es como sigue. Si pulsa **EDIT**, lo que se quiere borrar en la parte inferior de la pantalla se sustituirá por una copia de la línea en curso. Si pulsa ahora **ENTER**, dicha línea en curso volverá intacta al programa, dejando limpia la parte inferior de la pantalla.

Si introduce una línea por error, por ejemplo:

12 LET b=8

se incluirá en el programa y se apercibirá de su error. Para borrar esta línea innecesaria, teclee:

12 (con **ENTER,** por supuesto)

Observará con sorpresa que ha desaparecido el cursor del programa. Debe imaginar que está oculto entre las líneas 10 y 15, por lo que si pulsa ↑ se desplazará hasta la línea 10, mientras que si pulsa ↓ se desplazará hasta la línea 15.

Teclee:

12 (y **ENTER**)

De nuevo, el cursor del programa se ocultará entre las líneas 10 y 15. Ahora pulse **EDIT** y descenderá la línea 15; cuando el cursos del programa esté oculto entre dos líneas, **EDIT** bajará la siguiente línea después del nuevo número de línea. Teclee **ENTER** para borrar la parte inferior de la pantalla.

Ahora, teclee:

30 (y **ENTER**)

Esta vez, el cursor del programa está oculto tras el final del programa y si pulsa **EDIT**, la línea 20 bajará.

Finalmente, teclee:

LIST 15

Observará ahora en la pantalla:

15 LET b=15
20 PRINT a+b

La línea 10 ha desaparecido de la pantalla, pero sigue estando en su programa (lo que puede comprobar pulsando **ENTER**)**.** Los únicos efectos de **LIST 15** son producir un listado que comienza en la línea 15 y poner el cursor del programa en la

línea 15. Si tiene un programa muy largo, entonces, **LIST** será probablemente una forma más útil de desplazar el cursor del programa que con las teclas ↑ y ↓.

Esto ilustra otro uso de números de línea y es que actúan como nombres para las líneas del programa de modo que pueda hacer referencia a los mismos, de forma bastante parecida a cuando las variables tienen nombres.

LIST, sin ningún número, lista el programa a partir de la primera línea.

Otro comando visto en el fascículo de introducción es:

NEW

Su efecto es borrar tanto los programas como las variables que hubiese almacenados en el ordenador. Ahora, introduzca cuidadosamente este programa que cambia las temperaturas en grados Fahrenheit a grados Centígrados.

```
10 REM conversión de temperaturas
20 PRINT «grados F» , «grados C»
30 PRINT
40 INPUT F,(F-32)*5/9
50 GO TO 40
```

Necesitará teclear las palabras contenidas en la línea 10. Asimismo, aunque **GO TO** tiene un espacio entre ambos vocablos, todo es realmente una sola palabra clave (en **G**).

Ahora, haga la ejecución correspondiente. Verá los encabezamientos impresos en la pantalla por la línea 20, ¿pero qué sucedió con la línea 10? Aparentemente, el ordenador la ha ignorado completamente. En efecto, así es. **REM** en la línea 10 significa que se trata de una observación, o de un recordatorio, y su única misión es recordarle lo que hace el programa. Un comando **REM** está constituido por **REM seguido por algo que desee indicar y el ordenador lo Ignorará precisamente hasta el final de la línea.**

Hasta ahora, el ordenador ha llegado al comando **INPUT** en la línea 40 y está a la espera de que introduzca por el teclado un valor para la variable F (puede decirlo porque en donde podría haber esperado un cursor **K** hay, en su lugar, un cursor **L**). Introduzca un número; no se olvide de ENTER. Ahora, el ordenador ha presentado el resultado y está esperando otro número. Ello es así porque la línea 60, **GO TO 40** significa exactamente que el ordenador en vez de ejecutar el programa y detenerse, ha de saltar a la línea 40 y comenzar de nuevo. Por consiguiente, introduzca otra temperatura.

Después de introducir unos cuantos valores más podría llegar a preguntarse si la máquina no llegará a «aburrirse» realizando siempre la misma operación. Realmente no será así, pero la siguiente vez que le pida otro número puede teclear **STOP**. Entonces, el ordenador presentará un informe **D STOP in INPUT in line 40:1,** que le indica por qué se ha detenido y en dónde lo ha hecho (en el primer comando de la línea 40).

Si quiere continuar el programa, teclee:

CONTINUE

y el ordenador le pedirá otro número.

Cuando se utiliza **CONTINUE**, el ordenador recuerda el número de línea en el último informe que le transmitió, mientras no fuera **0 OK,** y salta a dicha línea; en nuestro caso, ello lleva consigo el salto a la línea 40, al comando INPUT.

Sustituya la línea 60 por **GO TO 31**, lo que no dará lugar a ninguna diferencia perceptible en la ejecución del programa. Si el número de línea en un comando **GO TO** se refiere a una línea no existente, entonces, el salto será a la siguiente línea después del número dado. Lo mismo es válido para **RUN**; de hecho, **RUN** en sí mismo significa realmente **RUN 0**.

Ahora introduzca por teclado números hasta que la pantalla comience a llenarse. Cuando esté llena, el ordenador desplazará el conjunto de la mitad superior de la pantalla hacia arriba una línea para hacer espacio, perdiendo el encabezamiento al salir por la parte superior. Este desplazamiento se denomina «scrolling».

Cuando se canse de realizar esta operación, detenga el programa con el empleo de **STOP** y consiga el listado pulsando **ENTER**.

Examine la sentencia **PRINT** en la línea 50. La puntuación en esta sentencia (la coma en este caso) es muy importante y debe tener presente que sigue reglas mucho más definidas que la puntuación en el idioma inglés.

Las comas se utilizan para hacer que la impresión comience en el margen izquierdo, o en la parte central de la pantalla, dependiendo de lo que venga a continuación. Así, en la línea 50, la coma hace que la temperatura centígrada se imprima en medio de la línea.- Por el contrario, con punto y coma, el siguiente número o cadena se imprime inmediatamente después de lo que le precede. Puede constatarlo en la línea 50, si sustituye la coma por punto y coma.

Otro signo de puntuación que puede utilizar como tal en los comandos **PRINT** es el apóstrofe ('). Este signo hace que lo que se imprima a continuación aparezca al principio de la siguiente línea en la pantalla, pero ello sucede, de todos modos, al final de cada comando **PRINT**, por lo que no tendrá mucha necesidad del apóstrofe. Esta es la razón por lo que el comando **PRINT** en la línea 50 comienza siempre su impresión en una línea nueva y también por qué el comando **PRINT** en la línea 30 produce una línea en blanco.

Si quiere inhibir lo anterior, de modo que después de un comando **PRINT** el siguiente quede en la misma línea, puede poner una coma, o un punto y coma, al final del primer comando. Para constatarlo, sustituya la línea 50 sucesivamente por cada una de las sentencias.

50 PRINT F,
50 PRINT F;

y

50 PRINT F

y ejecute cada variante. Para completarlo, podría probar también

50 PRINT F

El comando con la coma extiende todo en dos columnas, con el punto y coma presenta todo junto, sin ninguno de los dos signos anteriores asigna una línea para

cada número y lo mismo sucede con el apóstrofe (el apóstrofe da una nueva línea por sí mismo, pero inhibe la automática).

Tenga presente la diferencia entre las comas y el punto y coma en los comando **PRINT**; además, no debe confundirlos con los dos puntos (**:**) que se utilizan para separar comandos en una sola línea.

Ahora, teclee estas líneas suplementarias:

100 REM este programa cortés recuerda su nombre
110 INPUT n$
120 PRINT «Hola»; n$;"!"
130 GO TO 110

Este es un programa independiente del anterior, pero puede mantener a ambos en el ordenador al mismo tiempo. Para ejecutar el nuevo programa, teclee:

RUN 100

Puesto que este programa espera que se introduzca una cadena en lugar de un número, imprime dos comillas. Esto le ayuda a saber que debe introducir. Pruébelo escribiendo cualquier cosa.

Otra vez volverá a encontrarse con dos comillas, pero no tiene que utilizarlas si no lo necesita. Pruebe esto, por ejemplo: suprima las comillas (con □ y **DELETE)** y teclee:

n$

Puesto que no hay comillas de cadena, el ordenador sabe que ha de realizar algún cálculo; en este caso, el calculo consiste en determinar el valor de la variable de cadena denominada n$, que es cualquier nombre que casualmente haya introducido por el teclado en la última ocasión. Por supuesto, la sentencia **INPUT** actúa como **LET n$=n$,** por lo que se mantiene invariable el valor de n$.

A continuación, con fines comparativos, teclee:

n$

de nuevo, pero esta vez sin suprimir las comillas de cadena. Ahora, precisamente para confundirle, la variable n$ tiene el valor «n$».

Si desea utilizar **STOP** para la introducción de datos, debe desplazar primero el cursor haciéndolo retroceder al principio de la línea, con el empleo de la tecla □.

Volvamos, ahora, a ese **RUN 100** que teníamos anteriormente y que saltaba a la línea 100. ¿Podríamos haber puesto **GO TO 100** en su lugar? En este caso, hemos de dar una respuesta afirmativa en primera instancia, pero hay una diferencia. **RUN 100,** ante todo, suprime todas las variables y borra la pantalla y después de ello opera lo mismo que **GO TO 100.** Esta última no borra nada. Pueden darse casos en los que quiera ejecutar un programa sin borrar ninguna variable; entonces, sería necesaria **GO TO** y sin embargo, **RUN** podría tener efectos desastrosos. Por ello es preferible dejar la costumbre de teclear automáticamente **RUN** para ejecutar un programa.

Otra diferencia es que puede teclear **RUN** sin un número de línea y el ordenador

comienza en la primera línea del programa. **GO TO** debe tener siempre un número de línea.

Ambos programas se interrumpieron porque tecleó **STOP** en la línea de entrada; a veces, por error, escribe un programa que no puede detener y que no se parará por sí mismo. Teclee:

200 GO TO 200
RUN 200

Este programa tiene el «aspecto» de proseguir indefinidamente a no ser que se decida a desenchufar el aparato, pero hay un remedio menos drástico. Pulse **CAPS SHIFT** junto con la tecla **SPACE**, que tiene escrito **BREAK** por encima. El programa se interrumpirá, con el mensaje **L BREAK Into program** (interrupción del programa por ruptura).

Al final de cada sentencia, el programa constata si están pulsadas estas teclas y si lo están, el ordenador se para. La tecla **BREAK** puede emplearse también cuando esté utilizando el magnetofón, la impresora o cualquier otro dispositivo que pueda conectar al ordenador (precisamente en caso de que el ordenador esté esperando que ellos hagan algo, pero no lo hacen).

En estos casos, hay un informe diferente **D BREAK-CONT repeats.** En tales circunstancias, **CONTINUE** repite (y, de hecho, en la mayoría de los casos también) la sentencia en donde se detuvo el programa; pero después del informe **L BREAK into program, CONTINUE** pasa directamente a la siguiente sentencia después de permitir que se hagan algunos saltos.

Vuelva a ejecutar el programa del nombre y cuando el ordenador le pida que haga la introducción, teclee:

n$ (después de quitar las comillas)

n$ es una variable no definida y obtiene un informe de error **2 Variable not Found** (variable no encontrada). Si, ahora, teclea:

LET n$ = **«algo determinado»**

(que tiene su propio informe de **0 OK, 0:1)** y **CONTINUE**

encontrará que puede utilizar n$ como introducción de datos sin ninguna dificultad.

En este caso, **CONTINUE** realiza un salto al comando **INPUT** en la línea 110. No hace caso del informe de la sentencia **LET**, porque dijo «OK», y salta al comando al que se hizo referencia en el anterior informe, el primero en la línea 110. Se pretende que esto sea de utilidad. Si un programa se interrumpe por algún error, entonces puede hacer todo lo preciso para subsanar la anomalía y **CONTINUE** podrá operar todavía en lo sucesivo.

Como dijimos anteriormente, el informe **L BREAK into program** es especial, porque después del mismo **CONTINUE** no repite el comando en donde se detuvo el programa.

Los listados automáticos (los que no son el resultado de un comando **LIST**, sino que se producen después de introducir una nueva línea) quizás le hayan desconcer-

tado. Si introduce por el teclado en un programa con 50 líneas todas las sentencias **REM**

```
 1 REM
 2 REM
 3 REM
:   :
:   :
49 REM
50 REM
```

entonces será capaz de experimentar.

Lo primero a recordar es que la línea en curso (con **>**) siempre aparecerá en la pantalla y normalmente cerca de la parte central.

Pulse:

LIST (y **ENTER,** por supuesto)

y cuando el ordenador le pregunte **scroll?** (porque ha llenado la pantalla), pulse **n** para indicar «No». El ordenador emitirá el informe **D BREAK-CONT repeats** como si hubiera tecleado **BREAK.** En alguna ocasión, podría averiguar lo que sucede si pulsa **y** en lugar de **n; n, SPACE** y **STOP** cuentan como «no», mientras que todo lo demás cuenta como «sí».

Ahora, vuelva a pulsar **ENTER** para obtener un listado automático deberá ver las líneas 1 a 22 en la pantalla. Ahora, teclee:

23 REM

obtendrá las líneas 2 a 23 en la pantalla; teclee:

28 REM

y obtendrá las líneas 7 a 28. (En ambos casos, al teclear una nueva línea, ha desplazado el cursor del programa y por ello se ha realizado un nuevo listado).

Quizás esto le parezca algo arbitrario. Realmente, trata de proporcionarle exactamente lo que necesita, aunque al ser los «humanos criaturas imprevisibles, no siempre el ordenador adivina correctamente.

El ordenador mantiene un registro no solamente de la línea en curso (la que ha de aparecer en la pantalla), sino también de la línea superior en la pantalla. Cuando intenta hacer un listado, lo primero que hace es comparar la línea superior con la línea en curso.

Si la línea superior viene detrás, entonces no resulta oportuno comenzar en este punto y por ello el ordenador utiliza la línea en curso para una nueva línea superior y realiza su listado.

De cualquier otro modo, su método es comenzar la ejecución del listado a partir de la línea superior y proseguir hasta que la línea en curso sea objeto de listado, con un desplazamiento hacia arriba («scrolling»-arrollamiento) en caso de necesidad. Sin embargo, primero efectúa un calculo aproximado para ver cuánto ocupa y si la respuesta es que tiene una magnitud excesiva, entonces el ordenador desplaza la línea superior hacia abajo para estar mucho más próxima a la línea en curso. Ahora, al haber determinado su línea superior, el ordenador comienza el listado a partir de la misma. Si, cuando llega al final del programa o al fondo de la pantalla, se ha

listado la línea en curso, entonces, se detiene el ordenador. De no ser así, se produce un desplazamiento hacia arriba hasta que la línea en curso esté en la pantalla y por cada línea adicional que sea objeto de listado, el ordenador desplaza la línea superior hacia abajo en una línea, de modo que la línea superior se desplace a las proximidades de la línea en curso.

Experimente con el desplazamiento de la línea en curso, tecleando

número de línea **REM**

LIST desplaza la línea en curso pero no la línea superior, por lo que los subsiguientes listados podrían ser distintos. Por ejemplo, teclee

LIST

para conseguir el listado y luego pulse, de nuevo, **ENTER** para hacer que la línea 0 sea la línea superior. Debe tener las líneas 1 a 22 en la pantalla. Teclee:

LIST 22

que le proporciona las líneas 22 a 43; cuando pulse **ENTER** de nuevo, vuelva a las líneas 1 a 22. Ello tiende a ser más útil para programas cortos que para largos. Utilizando el programa lleno de sentencias **REM** anterior, teclee:

LIST

y luego n cuando el ordenador le pregunte **scroll?** Ahora, teclee:

CONTINUE

En este caso, **CONTINUE** tiene un carácter un poco «caprichoso» porque se pone en blanco la parte inferior de la pantalla; pero puede restaurar la normalidad con **BREAK.** La razón es que **LIST** era el primer comando en la línea, por lo que **CONTINUE** repite este comando. Lamentablemente, el primer comando en la línea es ahora **CONTINUE**, propiamente dicho, por lo que el ordenador se estabiliza realizando la función **CONTINUE** una y otra vez hasta que lo detenga.

Puede variar esta circunstancia sustituyendo **LIST** por

: LIST

para lo que **CONTINUE** da **0 OK** (porque **CONTINUE** salta al segundo comando en la línea, que se toma para se su final) o

:: LIST

para lo que **CONTINUE** da **N Statement lost** - sentencia perdida (porque **CONTINUE** salta al tercer comando en la línea, ya que no existe).

Ya ha visto las sentencias **PRINT, LET, INPUT, RUN, LIST, GO TO, CONTINUE, NEW y REM** y todas ellas pueden utilizarse como comandos directos o en líneas de programa (ello es cierto para casi todos los comandos en el ZX Spectrum en su versión particular del BASIC. **RUN, LIST, CONTINUE** y **NEW** no suelen ser de mucho uso en un programa, pero se pueden utilizar.

Ejercicios

1. Ponga una sentencia LIST en un programa de modo que cuando lo ejecute haga automáticamente un listado.

2. Escriba un programa para introducir precios e imprimir el impuesto adeudado (al 15 por ciento). Ponga las sentencias **PRINT** de modo que el ordenador indique lo que va a hacer y le pregunte el precio de entrada con extravagante cortesía. Modifique el programa para que también pueda introducir la tasa fiscal (para permitir las tasas de crecimiento cero o los cambios futuros).

3. Escriba un programa para imprimir el total de los números que introduzca. (Sugerencia: Tome dos variables denominadas **total** (inicialice esta con valor `0`) e **item**. Introduzca **item**, súmela a **total**, imprima ambas y vuelva a hacerlo otra vez).

4. ¿Qué harían **CONTINUE** y **NEW** en un programa? ¿Puede imaginar alguna aplicación?

CAPITULO



Decisiones

Resumen
IF, STOP
= , < , > , <= , >= , <>

Todos los programas que hemos visto hasta ahora han sido bastante predecibles (han pasado a través de las instrucciones y han vuelto al principio de nuevo). Ello no es de mucha utilidad. En la práctica, el ordenador habría de tomar decisiones y actuar en consecuencia. La Instrucción utilizada tiene la forma **... IF** (si) algo es verdadero, o no verdadero, **THEN** (entonces) hacer algo más.

Por ejemplo, utilice **NEW** para borrar el anterior programa del ordenador e introduzca por el teclado y ejecute el siguiente programa (que evidentemente admite que jueguen dos personas).

```
10 REM Adivine el número
20 INPUT a: CLS
30 INPUT «Adivine el número», b
40 IF b = a THEN PRINT «Correcto»: STOP
50 IF b<a THEN PRINT «Demasiado bajo, pruebe
   de nuevo»
60 IF b>a THEN PRINT «Demasiado alto, pruebe
    de nuevo»
70 GO TO 30
```

Puede constatar que una sentencia **IF** adopta la forma:

IF condición **THEN ...**

en donde «...» significa una secuencia de comandos, separados por dos puntos, en la forma habitual. La condición es algo que ha de determinarse como verdadero o falso; si es verdadero, se ejecutan las sentencias en el resto de la línea después de **THEN**, en caso contrario, se saltan y el programa ejecuta la siguiente instrucción.

Las condiciones más sencillas comparan dos números o dos cadenas. Pueden probar si dos números son iguales o si uno es mayor que el otro y pueden probar si dos cadenas son iguales o (aproximadamente) si una está antes que la otra en el orden alfabético. Utilizan las relaciones = , < , > , <= , >= , y <> .

= significa «igual a». Aunque es el mismo símbolo que el = en un comando **LET**, se utiliza en un sentido bastante diferente.

< **(SYMBOL SHIFT** con **R**) significa «es menos que», por lo que:

 1 < 2
-2 < -1
-3 < 1

son todas relaciones verdaderas, pero

1 < 0
0 < -2

son falsas.

La línea 40 compara **a** y **b**. Si son iguales, entonces, el programa se Interrumpe por el comando **STOP**. El informe en la parte inferior de la pantalla **9 STOP, statement, 40:3** indica que la tercera sentencia, o comando, en la línea 40 hizo que se detuviera el programa, esto es, **STOP**.

La línea 50 determina si **b** es menor que **a** y la línea 60 si **b** es mayor que **a**. Si una de estas condiciones es verdadera, entonces, se imprime el comentario correspondiente y el programa sigue su camino a la línea 70 que comunica al ordenador que vuelva a la línea 30 y allí comience de nuevo.

El comando **CLS**, borrar pantalla, en la línea 20 era para impedir que la otra persona vea lo que está introduciendo.

El símbolo > **(SYMBOL SHIFT** con **T**) significa «es mayor que» y es lo mismo que < pero en sentido contrario. Puede recordar cuál es cuál porque el extremo aguzado apunta siempre al número que se supone que es más pequeño.

<= **(SYMBOL SHIFT** con **O**), que no ha de teclearse < seguido por = , significa «es menor que o igual a», por lo que es como < con la salvedad de que es verdadero incluso si los dos números son iguales; por consiguiente 2<=2 es verdadero, pero 2 < 2 es falso.

>= **(SYMBOL SHIFT** con **E**) significa que «es mayor que o igual a» y es análogo a > , con las salvedades antes indicadas.

<> **(SYMBOL SHIFT** con **W**) significa «distinto a», lo opuesto a =.

Los matemáticos suelen escribir <= , >= y < > en la forma □ , □ y □. Escriben también cosas como '2 <3 <4' para significar '2<3' y '3 <4', pero ello no es posible en BASIC.

Observación: En algunas versiones de BASIC, pero no en la del ZX Spectrum, la sentencia **IF** puede tener la forma

IF condición THEN número de línea

Ello significa lo mismo que:

IF condición THEN GO TO número de línea.

Ejercicios

1. Pruebe este programa:

10 PRINT «x»: STOP: PRINT «y»

Cuando lo ejecute, el ordenador visualizará **x** y se parará con el informe **9 STOP statement, 10:2.** Ahora, pulse:

CONTINUE

Podría esperar que se produjera un salto atrás al comando **STOP** (**CONTINUE** suele repetir la sentencia que se refiere en el informe). Sin embargo, en este caso, ello no sería de mucha utilidad, porque el ordenador volvería a detenerse sin visualizar y. Por consiguiente, todo está dispuesto para que después del informe 9 **CONTINUE** salte al comando después del comando **STOP**. Por ello, en nuestro ejemplo, después de **CONTINUE**, el ordenador imprime y llega al final del programa.

CAPITULO 4

Iteración con bucles

Resumen
FOR, NEXT
TO, STEP

Suponga que desea introducir cinco números y sumarlos juntos. Una forma (que no le recomendamos que haga la introducción por el teclado a no ser que sea «masoquista») es escribir:

```
 10 LET total =0
 20 INPUT a
 30 LET total = total + a
 40 INPUT a
 50 LET total = total + a
 60 INPUT a
 70 LET total = total + a
 80 INPUT a
 90 LET total = total + a
100 INPUT a
110 LET total = total + a
120 PRINT total
```

Este método no es una buena práctica de programación. Puede ser tolerable para cinco números, pero puede imaginar cuán tedioso sería el procedimiento para sumar diez números y en el caso de un centenar sería prácticamente imposible.

Mucho mejor es establecer una variable para contar hasta 5 y luego, interrumpir el programa como en el desarrollo siguiente (que sí le instamos a introducir por el teclado):

```
10 LET total =0
20 LET cuenta = 1
30 INPUT a
40 REM cuenta = número de veces que a se ha introducido hasta ahora
50 LET total = total + a
60 LET cuenta = cuenta + 1
70 IF cuenta <= 5 THEN GO TO 30
80 PRINT total
```

Observe cuán fácil sería cambiar la línea 70 de modo que este programa sume diez números o incluso un centenar.

Esta clase de contaje es tan útil que hay dos comandos especiales para hacerlo más fácil: los comandos u órdenes **FOR** y **NEXT.** Siempre se utilizan conjuntamente.

Con el empleo de estos nuevos medios, el programa que acaba de introducir por el teclado realiza exactamente lo mismo que:

```
10 LET total = 0
20 FOR c=1 TO 5
30 INPUT a
40 REM c=número de veces que a se ha introducido hasta ahora
50 LET total = total + a
60 NEXT c
80 PRINT total
```

(Para conseguir este programa a partir del anterior, ha de corregir las líneas 20, 40, 60 y 70, **TO** es **SYMBOL SHIFT** con **F**).

Observe que hemos cambiando **cuenta** por **c**. La variable de contaje (o variable de control) de un bucle **FOR-NEXT** debe tener una sola letra para su nombre.

El efecto de este programa es que e opera a través de los valores 1 (el valor *inicial), 2,* 3, 4 y 5 (el *limite) y* para cada uno, se ejecutan las líneas 30, 40 y 50. A continuación, cuando **c** ha terminado con sus cinco valores, se ejecuta la línea 80.

Una sutileza adicional es que la variable de control no tiene que Incrementarse en 1 en cada ocasión, sino, que puede cambiar este 1 a cualquier otro valor que desee mediante el empleo de una parte **STEP** en el comando **FOR**. La forma más general para un comando **FOR** es:

FOR variable de control = valor inicial **TO** limite **STEP** paso

en donde la variable de control es una sola letra y el valor inicial, limite y paso son, en su totalidad, elementos que puede calcular el ordenador como números (como los números reales, o sumas o los nombres de variables numéricas. Por consiguiente, si sustituye la línea 20 en el programa por:

```
20 FOR c =1 TO 5 STEP 3/2
```

entonces **c** operará con los valores 1, 2’5 y 4. Observe que no tiene que limitarse a números enteros y también, que la variable de control no tiene que alcanzar el límite exactamente (mantiene la iteración por bucle en tanto que sea inferior o igual al límite).

Pruebe el siguiente programa para imprimir los números desde 1 a 10 en orden inverso.

```
10 FOR n =10 TO 1 STEP -1
20 PRINT n
30 NEXT n
```

Anteriormente dijimos que el programa mantiene la iteración por bucle mientras que la variable de control sea inferior o igual al límite. Si profundiza en lo que ello significaría en este caso, constará que no se trata de algo banal. La regla normal ha de modificarse cuando el paso es negativo pues, entonces, el programa realiza la iteración por bucle en tanto que la variable de control sea superior o igual al límite.

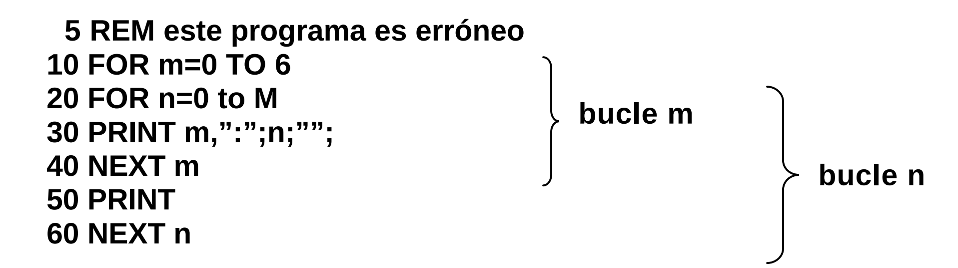
Debe tener cuidado si está realizando dos bucles **FOR-NEXT** conjuntamente, uno en el interior del otro. Pruebe el siguiente programa, que imprime los números correspondientes a un juego completo de dominó (de seis puntos).

```
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;":";n;"";
40 NEXT n
50 PRINT
60 NEXT m
```

bucle n (líneas 20–40)

bucle m (líneas 10–60)

Puede observar que el bucle **n** está completamente en el interior del bucle **m** (se dice que están adecuadamente *«anidados» o* encajados). Lo que debe evitarse es tener dos bucles **FOR-NEXT** que se solapen sin estar completamente uno en el interior del otro, como es el caso que se indica a continuación:



```
 5 REM este programa es erróneo
10 FOR m=0 TO 6
20 FOR n=0 to M
30 PRINT m,":";n;"";
40 NEXT m
50 PRINT
60 NEXT n
```


Dos bucles FOR-NEXT deben estar uno en el interior del otro o completamente separados.

Otra cosa a evitar es saltar a la parte media de un bucle **FOR-NEXT** desde el exterior. La variable de control sólo está adecuadamente establecida cuando ejecuta su sentencia **FOR** y si la omitiera, la sentencia **NEXT** produciría confusión en el ordenador. Probablemente obtendrá un informe de error con el mensaje **NEXT without FOR (NEXT** sin **FOR)** o **variable not found** (variable no encontrada).

No hay nada que le detenga con el empleo de **FOR** y de **NEXT** en un comando directo. Por ejemplo, pruebe:

```
FOR m=0 TO 10: PRINT m: NEXT m
```

A veces, puede utilizarle como una forma (algo artificial) de eludir la restricción de que no puede realizar la función **GO TO** en cualquier lugar en el interior de un comando (porque un comando no tiene ningún número de línea). Por ejemplo:

```
FOR m =0 TO 1 STEP 0: INPUT a: PRINT a: NEXT m
```

El paso (STEP) de cero, en este caso, hace que el comando (u orden) se repita a sí mismo «eternamente».

Este procedimiento no es recomendable, porque si se produce un error, entonces habrá perdido el comando y tendrá que volverle a introducir por el teclado (y **CONTINUE** ya no actuará).

Ejercicios

1. Una variable de control no sólo tiene un nombre y un valor, como una variable ordinaria, sino también un limite, un paso y una referencia a la sentencia después de la sentencia **FOR** correspondiente. Persuádase a sí mismo de que cuando se ejecuta la sentencia **FOR** toda esta información esté disponible (utilizando el valor inicial como el primer valor que toma la variable) y también de que esta información es suficiente para que la sentencia **NEXT** conozca en cuanto incrementar el valor, si ha de saltar hacia atrás y, de ser así, a dónde debe efectuarse el salto.

2. Ejecutar el tercer programa anterior y luego, teclee:

   PRINT c

   ¿Por qué la respuesta 6 y no 5?
   (Respuesta: El comando **NEXT** en la línea 60 se ejecuta cinco veces y cada vez se añade 1 a **c**. La última vez, **c** se hace 6 y luego, el comando **NEXT** decide no realizar el bucle hacia atrás, sino seguir adelante, al haber **c** sobrepasado su límite).
   ¿Qué sucede si pone **STEP 2** en la línea 20?

3. Cambie el tercer programa de modo que en lugar de sumar automáticamente cinco números, le pida que introduzca cuantos números quiera sumar. Cuando ejecute este programa, ¿qué sucede si introduce 0, con el significado de que no quiere sumar ningún número? ¿Por qué podría esperar que ello causara problemas al ordenador, aunque esté claro lo que usted quiere decir? (El ordenador ha de efectuar una búsqueda para el comando **NEXT c**, que no suele ser habitualmente necesario). De hecho, todo ello se ha tenido en cuenta.

4. En la línea 10 del cuarto programa anterior, cambie 10 por 100 y ejecute el programa. Imprimirá los números desde 100 a 89 en la pantalla y luego comunicará el mensaje **scroll?** en la parte inferior de la pantalla. Ello le da una oportunidad de ver los números que están a punto de desaparecer de la pantalla por la parte superior. Si pulsa **n**, **STOP** o la tecla **BREAK**, el programa se Interrumpirá con el informe **D BREAK-CONT repeats**. Si pulsa cualquier otra tecla, entonces el ordenador imprimirá otras 22 líneas y le interrogará de nuevo con la misma pregunta anterior.

5. Suprima la línea 30 del cuarto programa. Cuando ejecute el nuevo programa así abreviado, el ordenador imprimirá el primer número y se parará con el mensaje **0 OK.** Si teclea:

   NEXT n

   el programa recorrerá una vez el bucle e imprimirá el número siguiente.

CAPITULO 5

Subrutinas

Resumen
GO, SUB, RETURN

A veces, partes diferentes del programa tendrán tareas bastante similares que realizar y se encontrará tecleando las mismas líneas en dos o más ocasiones; sin embargo, ello no es necesario. Puede teclear las líneas una sola vez, en una forma conocida como una subrutina, y luego utilizarlas, o llamarlas, en cualquier lugar del programa sin necesidad de volverlas a introducir por el teclado.

Para realizar esta operación, utilice las sentencias **GO SUB (GO** a **SUB**rutina) y **RETURN**.

Ello adopta la forma:

GO SUB n

en donde n es el número de línea de la primera línea en la subrutina. Es lo mismo que **GO TO** n con la salvedad de que el ordenador recuerda en donde estaba la sentencia **GO SUB**, de modo que pueda retornar después de realizar la subrutina. Lo hace poniendo el número de línea y el número de la sentencia dentro de la línea (juntos constituyen la *dirección de retorno)* en la parte superior de una pila (la pila **GO SUB**).

RETURN

toma la dirección de retorno superior de la pila **GO SUB** y la lleva a la sentencia que le sigue.

A título de ejemplo, vuelva a examinar el programa para adivinar números. Vuelva a teclear lo siguiente:

```
 10 REM «Un juego de adivinación con nueva disposición»
 20 INPUT a: CLS
 30 INPUT «Adivine el número », b
 40 IF a = b THEN PRINT «Correcto»: STOP
 50 IF a<b THEN GO SUB 100
 60 IF a>b THEN GO SUB 100
 70 GO TO 30
100 PRINT «Pruebe de nuevo»
110 RETURN
```

La sentencia **GO TO** en la línea 70 es muy importante porque, de no ser por ella, el programa pasará en su ejecución a la subrutina y se producirá un error **(7 RETURN without GO SUB** - RETURN sin GO SUB) cuando se llegue a la sentencia **RE-TURN**.

Veamos otro programa bastante sencillo que ilustra el empleo de **GO SUB.**

```
100 LET x=10
110 GO SUB 500
120 PRINT s
130 LET x=x+4
140 GO SUB 500
150 PRINT s
160 LET x=x+2
170 GO SUB 500
180 PRINT s
190 STOP
500 LET s=0
510 FOR y=1 TO x
520 LET s=s+y
530 NEXT y
540 RETURN
```

Cuando se ejecute este programa, vea si puede determinar lo que está sucediendo. La subrutina comienza en la línea 500.

Una subrutina puede llamar a otra o incluso a sí misma (en este último caso, se llama **recursivo**).

CAPITULO



READ, DATA, RESTORE

Resumen
READ, DATA, RESTORE

En algunos programas anteriores, vimos que la información, o datos, puede introducirse directamente en el ordenador con el empleo de la sentencia **INPUT.** A veces, ello puede ser muy tedioso, sobre todo si muchos datos se repiten cada vez que se ejecuta el programa. Puede Ahorrar mucho tiempo utilizando los comandos **READ, DATA** y **RESTORE.** Por ejemplo:

```
10 READ a,b,c
20 PRINT a,b,c
30 DATA 10, 20, 30
40 STOP
```

Una sentencia READ está constituida por **READ** seguida de una lista de nombres de variables, separados por comas. Actúa de forma bastante similar a una sentencia **INPUT**, salvo que en lugar de hacerle introducir por el teclado los valores asignados a las variables, el ordenador busca los valores en la sentencia **DATA**.

Cada sentencia **DATA** es una lista de expresiones, numéricas o de cadenas, separadas por comas. Puede ponerlas en cualquier lugar que desee en un programa, porque el ordenador las ignora salvo cuando está realizando una sentencia **READ**. No obstante todas las sentencias **DATA** en el programa suelen ponerse juntas para formar una larga lista de expresiones, la *lista* de datos. La primera vez que el ordenador llega a la función **READ** de un valor, toma la primera expresión de la lista **DATA**; la siguiente vez, toma la segunda y, así, a medida que encuentra las sucesivas sentencias **READ**, abre su camino a través de la lista de DATA (si intenta ir más allá del final de dicha lista, se producirá un error).

Observe que es una pérdida de tiempo poner las sentencias **DATA** en un comando directo, porque **READ** no las encontrará. Las sentencias **DATA** han de ir en el programa.

Veamos cómo se encajan juntas en el programa que acaba de introducir por el teclado. La línea 10 dice al ordenador que efectúe la lectura de tres elementos de datos y los asigne a las variables **a, b** y **c**. La línea 20 le dice que imprima (**PRINT**) estas variables. La sentencia **DATA** en la línea 30, da los valores de **a, b** y **c**. La línea 40 detiene el programa. Para ver el orden en que se realizan las cosas, cambie la línea 20 a:

```
20 PRINT b,c,a
```

La información en **DATA** puede ser parte de un bucle **FOR ... NEXT**. Teclee.

```
10 FOR n=1 TO 6
20 READ D
30 DATA 2,4,6,8,10,12
40 PRINT D
50 NEXT n
60 STOP
```

Cuando este programa se ejecuta (**RUN**), puede ver a la sentencia **READ** desplazarse a través de la lista de **DATA**. Las sentencias **DATA** pueden contener también variables de cadena. Por ejemplo:

```
10 READ d$
20 PRINT «La fecha es», d$
30 DATA «1 junio 1982»
40 STOP
```

Esta es la forma sencilla de buscar expresiones a partir de la lista de **DATA:** Comenzar por el principio y seguir a través de ella hasta que llegue al final. Sin embargo, puede hacer que el ordenador salte a cualquier lugar en la lista de **DATA**, con el empleo de la sentencia **RESTORE**. Esta sentencia está constituida por **RESTORE** seguida por un número de línea y hace que las subsiguientes sentencias **READ** comiencen la lectura de sus datos a partir de la primera sentencia **DATA**, o después del número de línea dado (puede pasar por alto el número de línea, en cuyo caso es como si hubiera tecleado el número de la primera línea en el programa).

Pruebe este programa:

```
10 READ a,b
20 PRINT a,b
30 RESTORE 10
40 READ x,y,z
50 PRINT x,y,z
80 DATA 1,2,3
70 STOP
```

En este programa, los datos requeridos por la línea 10 hacen **a**=1 y **b**=2. La Instrucción **RESTORE 10** restaura las variables y permite que sean objeto de lectura (**READ**) las variables **x**, **y** y **z** comenzando a. partir del primer número en la sentencia **DATA**. Vuelva a ejecutar este programa sin la línea 30 y vea lo que sucede.

CAPITULO 7

Expresiones

Resumen
Operaciones: + , - , * , /.
Expresiones, notación científica, nombres de variables.

Ya se han visto algunas de las formas en que el ZX Spectrum puede calcular con números. Puede realizar las cuatro operaciones aritméticas + , **-** ,* y / (recuerde que ***** se utiliza para la multiplicación y ***/*** se emplea para la división) y puede determinar el valor de una variable, dado su nombre.

El ejemplo:

LET TAX =suma* 15/ 100

proporciona apenas una indicación del muy importante hecho de que pueden combinarse estos cálculos. Dicha combinación, como **suma*15/100,** se denomina una *expresión,* que es una forma abreviada de comunicar al ordenador que haga varios cálculos, uno después del otro. En nuestro ejemplo, la expresión **suma*15/100** significa «busque el valor de la variable denominada «suma», multiplíquele por 15 y divida por 100».

Si todavía no lo ha hecho, recomendamos que examine el fascículo de introducción para ver cómo el ZX Spectrum trabaja con los números y el orden en que evalúa el ordenador las expresiones matemáticas.

Para recapitular:

Las multiplicaciones y las divisiones se efectúan primero. Estas operaciones tienen una *prioridad más alta* que la suma y la resta. En relación entre ellas, la multiplicación y la división tienen la misma prioridad, lo que significa que ambas operaciones se efectúan en orden de izquierda a derecha. Una vez realizadas, se sigue con las sumas y las restas que también tienen la misma prioridad y que, por tanto, se efectúan en orden de izquierda a derecha.

Aunque todo lo que realmente necesita saber es si una operación tiene una prioridad más alta o más baja que otra, el ordenador realiza esto con la asignación de un número entre 1 y 16 para representar la prioridad de cada operación: * y / tienen prioridad 8 y + y - tienen prioridad 6.

Este orden de cálculo es absolutamente rígido, pero puede eludirlo con el empleo de paréntesis: cualquier cosa entre paréntesis se evalúa primero y luego se trata como un número único.

Las expresiones son útiles porque, siempre que el ordenador esté esperando que usted le introduzca un número, puede darle una expresión en lugar de tal número y el ordenador dará la solución. Las excepciones a esta regla son tan pocas que se establecerán explícitamente en cada caso.

Puede reunir tantas cadenas (o variables de cadena) como quiera en una sola expresión y si lo requiere, incluso puede utilizar paréntesis.

Realmente tenemos la obligación de decirle lo que puede y lo que no puede utilizar como nombre de variables. Como ya dijimos, el nombre de una variable de cadena ha de ser una sola letra seguida por $ y el nombre de la variable de control de un bucle **FOR-NEXT** debe ser una sola letra, pero los nombres de las variables numéricas ordinarias son mucho más libres. Puede utilizar ya sean letras o dígitos, siempre y cuando comiencen por una letra. Puede poner espacios si con ellos se fa-

cilita la lectura, pero no han de considerarse como partes del nombre. Asimismo, no se establece ninguna diferencia para el nombre si lo escribe en mayúsculas o en minúsculas.

Damos algunos ejemplos de los nombres de variables que están permitidos:

x
t42
este nombre es tan largo que nunca será capaz de volverlo a escribir sin cometer un error
ahora somos seis (estos dos últimos nombres se consideran los
aHORasoMosSEIS mismos y se refieren a la misma variable)

Lo que sigue no se permite como nombres de variables:

2001	(comienza con un dígito)
3 osos	(empieza con un dígito)
M*A*S*H	(* no es una letra ni un dígito)
Segado-Angel	(- no es una. letra ni un dígito)

Las expresiones numéricas pueden representarse por un número y un exponente (una vez más, haga referencia al fascículo de introducción). Pruebe lo siguiente para constatarlo:

```
PRINT 2.34e0
PRINT 2.34e1
PRINT 2.34e2
```

y así sucesivamente, hasta

```
PRINT 2.34e15
```

Observará que, después de unos instantes, el ordenador comienza también a utilizar la notación científica. Esta es la razón por la que no más de catorce caracteres pueden emplearse para escribir un número. Análogamente, pruebe:

```
PRINT 2.34e-1
PRINT 2.34e-2
```

y así sucesivamente.

PRINT da solamente ocho dígitos significativos de un número. Pruebe:

```
PRINT 4294967295, 4294967295-42907
```

Ello prueba que el ordenador puede retener los dígitos de 4294967295, aun cuando no esté preparado para visualizarlos todos a la vez.

El ZX Spectrum utiliza la aritmética de coma flotante, lo que significa que mantiene separados los dígitos de un número (su mantisa) y la posición del punto (el exponente). Ello no es siempre exacto, incluso para números enteros. Teclee:

```
PRINT 1e10+ 1-1e10,1e10-1e10+ 1
```

Los número se calculan con una precisión de 9 ½ dígitos, por lo que 1e10 es demasiado grande para ser almacenado con una precisión absolutamente correcta. La inexactitud (realmente casi 2) es superior a 1 y por ello los números 1e10 y 1e10+1 son para el ordenador aparentemente iguales.

Para un ejemplo todavía más peculiar, teclee:

PRINT 5e9+ 1-5e9

En este caso, la inexactitud en 5e9 es sólo de casi 1 y el 1 a añadir se redondea, por tanto, a 2. Los números 5e9+1 y 5e9+2 son para el ordenador aparentemente iguales.

El mayor número entero que puede obtenerse con exactitud completa 2^{32} - 1 (o 4.294.967.295).

La cadena “” sin ningún carácter en absoluto se denomina la cadena *nula o* vacía. Recuerde que los espacios son significativos y una cadena vacía no es lo mismo que una que no contenga nada sino espacios.

Pruebe:

PRINT «No has acabado «El Quijote» todavía?»

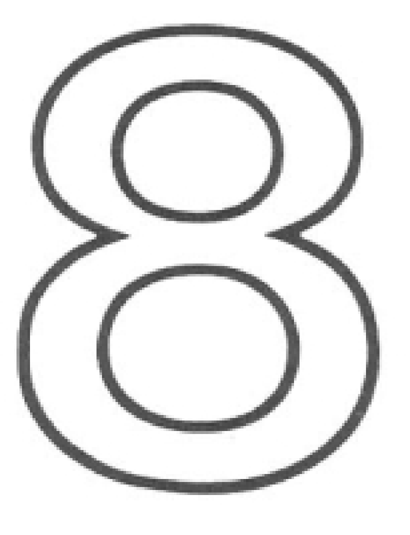
Cuando pulse ENTER, obtendrá el signo de interrogación parpadeante que indica que hay un error en algún lugar de la línea. Cuando el ordenador encuentra las dobles comillas al principio de «El Quijote», imagina que ellas marcan el final de la cadena «No has acabado» y, entonces, no puede determinar lo que significa «El Quijote».

Hay un subterfugio para superar este inconveniente: siempre que requiera escribir un símbolo de comillas de cadena en la parte media de una cadena, debe escribirlo por dos veces, como sigue:

PRINT «No has acabado ««El Quijote»» todavía?»

Como puede observar por lo que se imprime en la pantalla, cada doble comilla está realmente sólo una vez; las ha escrito dos veces simplemente para que el ordenador las identifique de forma adecuada.

CAPITULO



Cadenas

Resumen
Fragmentación, utilizando **TO**. Observe que esta notación no es de BASIC estándar.

Dada una cadena, una subcadena de la misma está constituida por algunos caracteres consecutivos de ella tomados en secuencia. Por consiguiente, «string» es una subcadena de «bigger string» (una cadena más grande), pero no son subcadenas «b sting» y «big reg».

Hay una notación denominada «slicing» (fragmentación) para describir las subcadenas y puede aplicarse a cualquier expresión de cadena. La forma general es:

expresión de cadena (comienzo **TO** final)

de modo que, por ejemplo:

«abcdef» (2 TO 5) = «abcde»

Si omite el comienzo, entonces se supone que es 1; si omite el final, entonces se supone la longitud de la cadena. Por consiguiente:

«abcdef »(TO 5) = «abcdef »(1 TO 5) = «abcde»
«abcdef »(2 TO) = «abcdef»(2 TO 6) = «bcdef»
«abcdef »(TO) = «abcdef »(1 TO 6) = «abcdef»

(también puede escribir la última como **«abcdef»()**, ya que lo admite.

Una forma algo diferente es cuando se omite **TO** y sólo hay un número.

«abcdef»(3)= «abcdef»(3 TO 3) = «c»

Aunque el comienzo y el final suelen referirse a partes existentes de la cadena, esta regla queda derogada por otra: si el valor es mayor que el contenido de la cadena, el resultado es una cadena vacía. Así:

«abcdef»(5 TO 7)

da el error **3 subscript wrong** porque la cadena sólo contiene 6 caracteres y 7 es superior, pero:

«abcdef »(8 TO 7) = ""

y

«abcdef»(1 TO 0) = ""

El comienzo y el final no deben ser negativos u obtendrá un error **B Integer out of range** (entero B fuera de margen). El siguiente ejemplo es una simple ilustración de estas reglas.

```
10 LET a$= «abcdef»
20 FOR n=1 TO 6
30 PRINT a$(n TO 6)
40 NEXT n
50 STOP
```

Teclee **NEW** cuando se haya ejecutado este programa e introduzca el programa siguiente:

```
10 LET a$ = «No lo creo»
20 FOR n =1 TO 10
30 PRINT a$(n TO 10),a$((10-n) TO 10)
40 NEXT n
50 STOP
```

Para variables de cadena, podemos no solamente extraer subcadenas, sino también asignarlas.

Por ejemplo, teclee:

```
LET a$ = «Yo soy el ZX Spectrum»
```

y luego:

```
LET a$(1 TO 3) = ”******”
```

y

```
PRINT a$
```

Observe cómo puesto que la subcadena **a$(5 TO 8)** sólo tiene 4 caracteres de longitud, se han utilizado únicamente las cuatro primeras estrellas. Esta es una característica de la asignación a subcadenas: la subcadena ha de tener exactamente la misma longitud después que antes de su asignación. Para cerciorarse de que sucede así, la cadena Implicada en la asignación se corta a la derecha si es demasiado larga o se rellena con espacios si es demasiado corta. Este método se llama asignación procusteana en rememoración del célebre posadero Procrustes que acostumbraba a cerciorarse de que sus huéspedes encajaban en la cama estirándoles en un potro de tormento o cortándole los pies.

Si, ahora, prueba:

```
LET a$() = «Cómo estás?»
```

y

```
PRINT a$;”.”
```

observará que vuelve a suceder lo mismo (esta vez con espacios introducidos) porque **a$()** cuenta como una subcadena

LET a$ = **«Cómo estás»?**

lo hará adecuadamente.

Las expresiones de cadenas complicadas necesitarán encerrarse entre paréntesis antes de que puedan fragmentarse. Por ejemplo:

«abc» + «def »(1 TO 2) = «abcde»
("abc" + "def")(1 TO 2) = "ab"

Ejercicio

1. Intente escribir un programa para que se imprima el día de la semana utilizando el método de la fragmentación.Recomendación: Se sugiere que la cadena sea DomLunMarMiercJuevVieSab.

CAPITULO



Funciones

Resumen
DEF
LEN, STR$, VAL, SGN, ABS, INT, SQR
FN

Considere una máquina de hacer embutidos. Suponga que pone un trozo grande de carne en un extremo, gira una manivela y obtiene a la salida en el otro extremo, un embutido de carne. Un hecho análogo se tendría con cualquier otro ingrediente destinado a la fabricación de embutido.

Las funciones son algo similar a estas máquinas de embutidos pero hay una diferencia: trabajan con números y cadenas de caracteres en lugar de hacerlo con carne. Suministre un valor (llamado el *argumento),* trátelo efectuando algunos cálculos y, finalmente, obtendrá otro valor, el *resultado.*

Entrada de carne → Máquina de embutidos → Salida de embutido

Entrada argumento → Función → Salida resultado

Argumentos diferentes proporcionan resultados diferentes y si el argumento es completamente incorrecto, la función se interrumpirá y dará un informe de error.

Lo mismo que puede tener máquinas diferentes para obtener productos diferentes, funciones distintas realizarán cálculos distintos. Cada una tendrá su propio valor para distinguirla de las demás.

En las expresiones, una función se utiliza escribiendo su nombre seguido del argumento y cuando la expresión es evaluada, se obtiene el resultado de la función.

Como ejemplo, hay una función denominada **LEN**, que da como resultado la longitud de una cadena de caracteres. Su argumento es la cadena cuya longitud queremos determinar y su resultado es la longitud, de modo que si teclea:

PRINT LEN "Sinclair"

el ordenador escribirá la respuesta 8, que es el número de letras existentes en la palabra «Sinclair». (Para obtener **LEN**, como en el caso de la mayoría de los nombres de funciones, debe utilizar el modo extendido, esto es: pulse **CAPS SHIFT** y **SYMBOL SHIFT**, al mismo tiempo, para cambiar el cursor de **L** a **E** y luego, pulse la tecla **L**.

Si mezcla funciones y operaciones en una sola expresión, entonces las funciones tendrán prioridad sobre las operaciones. Sin embargo, también en este caso, puede eludir esta regla utilizando paréntesis. Por ejemplo, damos a continuación dos expresiones que sólo difieren en los paréntesis y, sin embargo, los cálculos se realizan en un orden completamente distinto en cada caso (aunque, como por casualidad, los resultados finales son los mismos).

LEN "Fred" + **LEN** "Bloggs"	**LEN** ("Fred" + "Bloggs")
4+ LEN "Bloggs"	**LEN** ("FredBloggs")
4 + 6	**LEN** "FredBloggs"
10	10

Veamos, ahora, algunas otras funciones.

STR$ convierte números en cadenas: su argumento es un número y su resultado es la cadena que aparecía en la pantalla si el número se visualizara por medio de una sentencia **PRINT**. Observe cómo su nombre finaliza en un signo $ para indicas que su resultado es una cadena. Por ejemplo, podría decir:

LET a$ = STR$1e2

que tendría exactamente el mismo efecto que si se tecleara:

LET a$ = "100"

O también podría escribir:

PRINT LEN STR$ 100.0000

y obtener la respuesta 3, porque **STR 100.0000= "100"**.

VAL es similar a **STR$** en sentido inverso: convierte cadenas en números. Por ejemplo:

VAL "3.5" = 3.5

En un sentido, **VAL** es la inversa de **STR$**, porque si toma cualquier número, le aplica **STR$** y luego, le aplica **VAL**, volverá a obtener el número original.

Si toma una cadena, le aplica **VAL** y luego, le aplica **STR$**, no siempre volverá a su cadena original.

VAL es una función muy potente, porque la cadena en la que está su argumento no está restringida a que sea como un número sencillo, sino que puede ser cualquier expresión numérica. Así, por ejemplo:

VAL "2*3"=6

o también:

VAL ("2"+"*3”)=6

Hay dos procesos actuando en este caso. En el primero, el argumento de **VAL** se evalúa como una cadena: la expresión de cadena **"2"+"*3"** se evalúa para obtener la cadena **"2*3"**. Entonces, se eliminan las comillas de la cadena y lo que queda se evalúa como un número: así pues, **2*3** una vez avaluado dará el número 6.

Esto puede ser extremadamente confuso si no conserva su «presencia de ánimo»; por ejemplo:

PRINT VAL "VAL""VAL “”””2”””””””

(Recuerde que en el interior de una cadena unas comillas de cadena deben escribirse dos veces. Si profundiza más en las cadenas, encontrará que las comillas de cadena necesitan cuadruplicarse o incluso octuplicarse).

Hay otra función, bastante similar a **VAL**, aunque probablemente menos útil, denominada **VAL$**. Su argumento sigue siendo una cadena, pero su resultado es también una cadena. Para ver cómo actúa, recuerde que **VAL** tiene una acción en en dos pasos: primero, su argumento se evalúa como un número. Con **VALS$**, el primer paso es el mismo, pero después de eliminarse las comillas de cadena en el segundo paso, lo que queda se evalúa como otra cadena. Así:

VAL$"""Chocolate""" = "Chocolate"

(Observe cómo las comillas de cadena proliferan de nuevo).

Haga:

LET a$ = "99"

e Imprima todo lo siguiente: **VAL a$, VAL "a$", VAL """ a$""", VAL$ a$, VAL "a$"** y **VAL$ """a$"""$"""**. Algunas de ellas actuarán y otras no lo harán; intente encontrar explicación a todas las respuestas.

SGN es la función *signo.* Es la primera función que ha visto que no tiene nada que hacer con las cadenas, porque su argumento y su resultado son números. El resultado es + 1 si el argumento es positivo, 0 si el argumento es cero y -1 si el argumento es negativo.

ABS es otra función cuyo argumento y cuyo resultado son números. Convierte el argumento en un número positivo (que es el resultado) haciendo caso omiso del signo, de modo que por ejemplo:

ABS -3.2 = ABS 3.2 = 3.2

INT significa «parte entera» (un *entero* es un número entero, que puede ser negativo). Esta función convierte un número fraccionario, con cifras decimales, en un entero suprimiendo la parte fraccionaria, así por ejemplo:

INT 3.9 = 3

Ha de tener cuidado al aplicar esta función a números negativos, porque siempre hay que redondear por defecto; así, por ejemplo:

INT - 3* = - 4

SQR calcula la *raíz cuadrada* de un número y el resultado multiplicado por sí mismo, da el argumento. Por ejemplo:

SQR 4 = 2 porque 2*2 x 4
SQR 0.25 = 0.5 porque 0.5*0.5=0.25
SQR 2 = 1.4142136 (aproximadamente)
porque 1.4142136*1.4142136 = 2.0000001

Si multiplica cualquier número (incluso uno negativo) por sí mismo, la respuesta es siempre positiva. Ello significa que los números negativos no tienen raíces cuadradas, por lo que si aplica **SQR** a un argumento negativo obtendrá un informe de error **An Invalid Argument** (un argumento no válido).

Asimismo, puede definir funciones por su cuenta. Posibles nombres para estas funciones son **FN** seguida por una letra (si el resultado es un número) o **FN** seguida por una letra y por $ (si el resultado es una cadena). Hay más rigurosidad con respecto a los paréntesis: el argumento debe encerrarse entre paréntesis.

Defina una función poniendo una sentencia **DEF** en algún lugar del programa. Por ejemplo, veamos la definición de una función **FN s**, cuyo resultado sea el cuadrado del argumento:

```
10 DEF FN s(x) =x*x: REM el cuadrado de x
```

DEF se obtiene en modo extendido, con el empleo de **SYMBOL SHIFT** y de **1**. Cuando haya escrito esto, el ordenador le dará automáticamente **FN**, porque en una sentencia **DEF, FN** siempre va inmediatamente detrás de **DEF**. A continuación, la letra N completa el nombre **FN s** de la función.

La **x** entre paréntesis es un nombre mediante el cual desea referirse al argumento de la función. Puede utilizar cualquier letra única que quiera para ello (o, si el argumento es una cadena, una sola letra seguida por $).

Después del signo = está la definición real de la función. Esta puede ser cualquier expresión y puede referirse también al argumento utilizando el nombre que le ha dado (en este caso, **x**) como si fuera una variable ordinaria.

Una vez introducida esta línea, puede invocar, o llamar, la función como si se tratara de las propias funciones del ordenador, tecleando su nombre, **FN s**, seguido por el argumento. Recuerde que una función definida por usted, debe llevar el argumento encerrado entre paréntesis. Pruébelo unas cuantas veces:

```
PRINT FN s(2)
PRINT FNs(3 + 4)
PRINT 1 + INT FN s (LEN "pollo"/2+ 3)
```

Una vez que haya puesto la sentencia **DEF** correspondiente en el programa, puede utilizar sus propias funciones en expresiones con la misma libertad que puede emplear las del ordenador.

Nota: En algunas versiones de BASIC, debe encerrar entre paréntesis el argumento de una de las funciones del ordenador. Este no es el caso en el BASIC del ZX Spectrum.

INT siempre, redondea por defecto. Para redondear al entero más próximo., añada primero 0.5 (podría escribir su propia función para hacer esta operación).

```
20 DEF FN r(x) = INT (x+0.5): REM da x redondeada al entero
   más próximo.
```

Entonces, obtendrá, por ejemplo:

FN r(2.9) = 3 **FN r(2.4) = 2**
FN r(-2.9) = -3 **FN r(-2.4) = -2**

Compare estas respuestas con las que consigue cuando utiliza **INT** en lugar de **FN r**. Escriba y ejecute lo siguiente:

```
10 LET x=0 : LETy=0 : LET a=10
20 DEF FN p(x,y)=a+x*y
30 DEF FN q()=a+x*y
40 PRINT FN p(2,3), FN q()
```

Hay muchos puntos sutiles en este programa.

Primero, una función no está restringida a un solo argumento: puede tener más o incluso ninguno en absoluto, pero siempre debe mantener los paréntesis.

Segundo, no importa en qué lugar del programa ponga las sentencias **DEF**. Una vez que el ordenador haya ejecutado la línea 10, simplemente salta por encima de las líneas 20 y 30 para llegar a la línea 40. Sin embargo, han de estar en cualquier lugar del programa y no pueden estar en un comando.

Tercero, **x** e **y** son los nombres de variables en el programa como un conjunto, y los nombres de argumentos para la función **FN p**. **FN p** olvida temporalmente las variables denominadas **x** e **y**, pero puesto que no tiene ningún argumento denominado **a**, todavía recuerda la variable **a**. Por consiguiente, cuando se está evaluando **FN p(2,3)**, **a** tiene el valor 10 porque es la variable, **x** tiene el valor 2 porque es el primer argumento **e** y tiene el valor 3 porque es el segundo argumento. El resultado es, entonces: 10+2*3=16. Cunado se está evaluando **FN q()**, por el contrario, no hay ningún argumento, por lo que **a**, **x** e **y** se refieren todavía a las variables y tienen los valores 10, 0 y 0 respectivamente. La respuesta, en este caso, es:
10 + 0*0 =10.

Ahora cambie la línea 20 a:

```
20 DEF FN p(x,y)= FN q()
```

Esta vez, **FN p(2,3)** tendrá el valor 10 porque **FN q** volverá también a las variables **x** e **y** en lugar de emplear los argumentos de **FN p**.

Algunas versiones del BASIC (y no precisamente la del BASIC del ZX Spectrum) tienen funciones denominadas LEFT$; RIGHT$, MID$ y TL$.

LEFT$ (a$,n) da la subcadena de a$ que consta de los primeros n caracteres.

RIGHT$ (a$,n) da la subcadena de a$ que está constituida por los caracteres desde el enésimo en adelante.

MID$ (a$, n1, n2) da la subcadena de a$ que está constituida por n2 caracteres que comienzan en el *n*1-ésimo.

TL$ (a$) da la subcadena de a$ que está constituida por todos sus caracteres con la excepción del primero.

Puede escribir algunas funciones definidas por el usuario para hacer lo mismo, por ejemplo:

```
10 DEF FN t$(a$)=a$(2 TO): REM TL$
20 DEF FN 1$(a$,n) =a$ (TO n): REM LEFT$
```

Compruebe que estas actúan con cadenas de longitud 0 y 1.

Observe que nuestra función **FN l$** tiene dos argumentos, uno es un número y el otro es una cadena. Una función puede tener hasta 26 argumentos numéricos (¿por qué precisamente 26?) y, al mismo tiempo, hasta 26 argumentos de cadena.

Ejercicio

Utilice la función **FN s(x)=x*x** para probar **SQR**; debe encontrar que

FN s(SQR x) = x

si sustituye **x** por cualquier número positivo y

SQR FN s(x) = ABS x

si **x** es un número positivo o negativo (¿por qué **ABS**?).

CAPITULO 10

Funciones matemáticas

Resumen
↑
PI; EXP, LN, SIN, COS, TAN, ASN, ACS, ATN

Este capítulo trata de las funciones matemáticas de que dispone del ZX Spectrum. Es bastante probable que nunca tenga que hacer uso de estas funciones en absoluto, por lo que si lo encuentra demasiado pesado, no vacile en hacer caso omiso de este capítulo. Abarca la operación ↑ (elevación a una potencia), las funciones **ESP** y **LN** y las funciones trigonométricas **SIN, COS, TAN** y sus inversas **ASN**, **ACS** y **ATN**.

↑ y EXP

Puede elevar un número a la potencia de otro, ello significa «multiplicar el primer número por sí mismo el número de veces indicado por el segundo». Esto se suele indicar escribiendo el segundo número inmediatamente encima y a la derecha del primer número; pero, evidentemente, esta notación sería difícil en un ordenador y por ello utilizamos el símbolo T en su lugar. Por ejemplo, las potencias de 2 son:

2 ↑ 1 = 2
2 ↑ 2 = 2*2 = 4 (2 al cuadrado, normalmente escrito 2)
2 ↑ 3 = 2*2*2 = 8 (2 al cubo, normalmente escrito 2)
2 ↑ 4 = 2`2'2'2 = 16 (2 a la cuarta potencia, normalmente escrito 2)

Por consiguiente, a su nivel más elemental, 'a ↑ *b'* significa 'a multiplicada por sí misma *b* veces', pero, evidentemente, esta operación sólo tiene sentido si b es un número entero positivo. Para encontrar una definición que sirva para otros valores de *b,* consideramos la regla:

a ↑ (b+c) = a↑b * a↑c

(Observe que damos a Tuna prioridad mayor que a * y /*,* por lo que cuando haya varias operacions en una sola expresión, las operaciones ↑ se efectúan antes de las operaciones * y /). Es bastante claro que lo anterior se verifica cuando *b y c* son números enteros positivos; pero si decidimos que necesitamos operar incluso cuando no lo son, entonces, nos veremos obligados a aceptar que:

a ↑ 0 = 1
a ↑ (-b) = 1/a ↑ b
a ↑ (1/b) = la raíz b-ésima de *a*, o lo que es lo mismo, el número que tiene que multiplicar por sí mismo *b* veces para obtener *a*

y

a ↑ (b*c) = (a ↑ b) ↑ c

Si nunca vio alguna de estas expresiones con anterioridad, entonces, no trate de recordarlas directamente; limítese a recordar que:

a ↑ (-1) = 1/a
y
a ↑ (1/2) = **SQR** (a)

y quizás cuando esté familiarizado con estas expresiones, el resto comenzará a tener sentido.

Experimente con todo ello probando este programa:

```
10 INPUT a,b,c,
20 PRINT a ↑ (b+c)=a ↑ b * a ↑ c
30 GO TO 10
```

Por supuesto, si la regla que dimos anteriormente es cierta, los números que imprima cada vez el ordenador serán iguales. (Observación: Habida cuenta de la forma en que el ordenador actúa con respecto a la operación ↑ **,** el número a la izquierda, que es **-a** en este caso, nunca debe ser negativo).

Un ejemplo bastante típico de la posible aplicación de esta función es la del interés compuesto. Suponga que tiene invertido parte de su dinero en una sociedad inmobiliaria y ésta le devenga un interés anual del 15%. Entonces, transcurrido un año, no sólo tendrá el 100% que tendría de todos modos, sino también los intereses del 15% que la sociedad inmobiliaria le haya proporcionado, lo que totaliza el 115% respecto a la cantidad que tenía originalmente. Para exponerlo de otra forma, ha multiplicado su cantidad de dinero por un factor de 1.15 como consecuencia de esta operación. Transcurrido otro año, volverá a suceder lo mismo, por lo que tendrá 1.15*1.15 = 1.15↑2 =1.3225 veces su cantidad de dinero original. En general, transcurridos **y** años, tendrá **1.15 ↑ y** veces la cantidad inicial de que disponía.

Si prueba este comando:

```
FOR y =0 TO 100: PRINT n, 10,10*1.15↑y: NEXT y
```

verá que incluso partiendo de solamente 2.000 pesetas su montante crece bastante rápidamente y, lo que es más, crece con mayor rapidez a medida que transcurre el tiempo (aún siendo así, podría considerar que no se mantiene a la altura de la inflación).

Esta clase de comportamiento, en donde transcurrido un intervalo de tiempo fijo esa cantidad *se* multiplica por sí misma en una proporción fija, se denomina *crecimiento exponencial, y* se calcula elevando un número fijo a la potencia del tiempo.

Suponga que hizo lo siguiente:

```
10 DEF FN a(x)=a ↑ x
```

En este caso, *a* es más o menos fija, mediante sentencias **LET**; su valor corresponderá al tipo de interés, que sólo cambia periódicamente.

Hay un determinado valor para a que hace a la función **FN a** especialmente atractiva a los ojos de un matemático y este valor se denomina *e*. El ZX Spectrum tiene una función denominada **EXP** definida por:

```
EXP x = e↑x
```

Lamentablemente, *e* en sí mismo no es un número especialmente atractivo, pues se trata de un número decimal no recurrente con infinitas cifras. Puede ver sus primeras cifras decimales haciendo

PRINT EXP 1

porque **EXP 1= e↑1 =e**. Por supuesto, esto es sólo una aproximación. Nunca puede obtener el número e con exactitud.

LN

La inversa de una función exponencial es una función *logarítmica;* el logaritmo (en base *a*) de un número *x* es la potencia a la que tiene que elevar a para obtener el número *x* y se escribe $\log_a x$. Así, por definición, a↑ $\log_a x$ =x y es también cierto que log (a↑x) =x.

Puede que ya conozca cómo utilizar los logaritmos de base 10 para efectuar multiplicaciones (dichos logaritmos se denominan logaritmos *vulgares).* El ZX Spectrum tiene una función **LN** que calcula logaritmos en base *e* (que se conocen como logaritmos *naturales).* Para calcular logaritmos en cualquier otra base, debe dividir el logaritmo natural por el logaritmo natural de la base:

$\log_a x$ = **LN x / LN a**

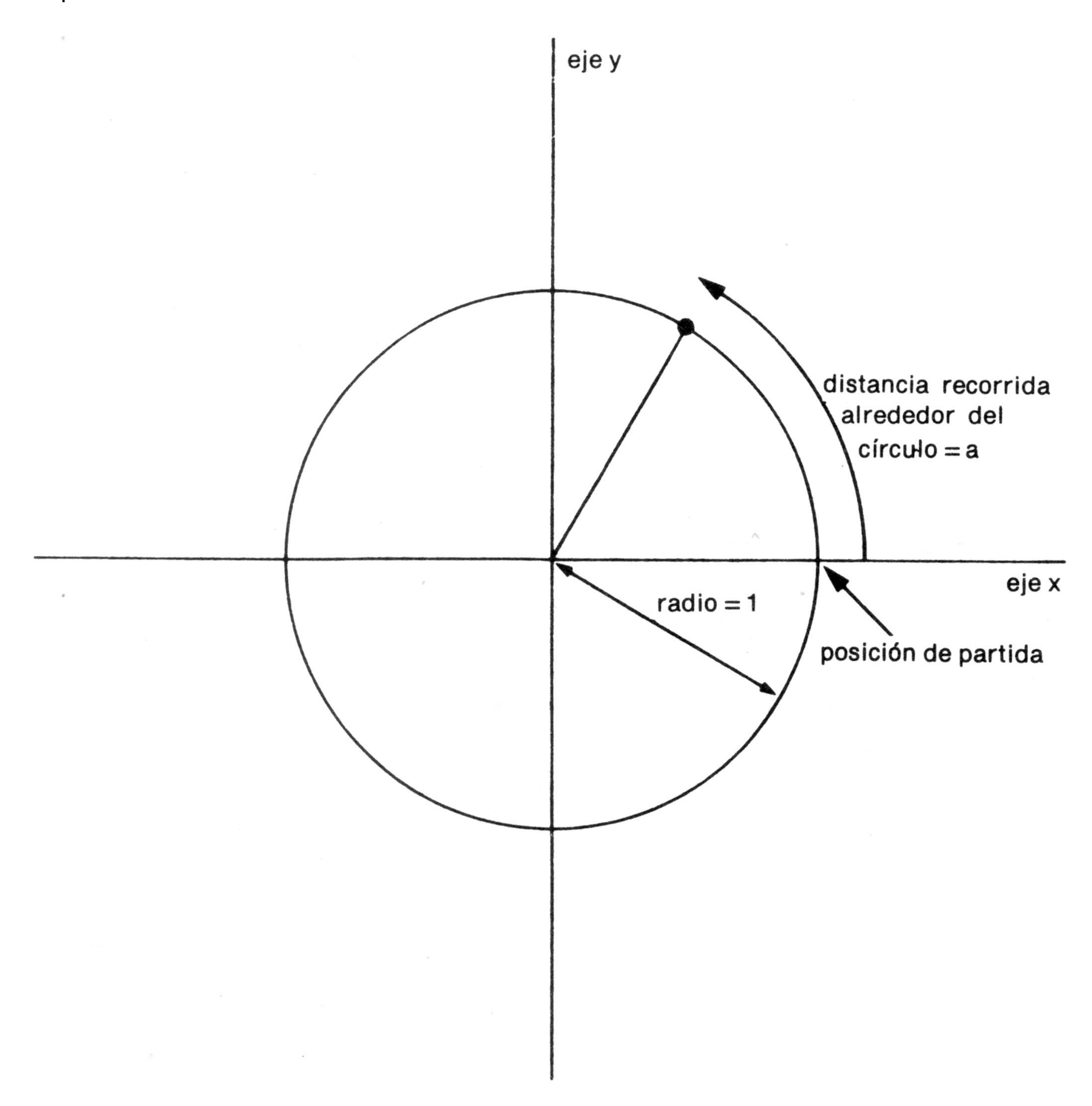
PI

Dado cualquier círculo, puede determinar su *perímetro* (la longitud de su contorno, que suele denominarse su *circunferencia)* multiplicando su *diámetro* (anchura) por un número llamado π (letra griega que significa perímetro).

Como el número *e*, π es un decimal no periódico y su valor es 3.141592653589... La palabra **PI** en el Spectrum (en el modo extendido, entonces, M) se toma con el significado de este número. Pruebe **PRINT PI.**

SIN, COS y **TAN, ASN, ACS** y **ATN**

Las funciones *trigonométricas* miden lo que sucede cuando un punto se desplaza alrededor de un círculo. Consideremos un círculo de radio 1 (este valor del radio no es crítico y se toma para facilitar los cálculos) y un punto que se desplaza alrededor del mismo. El punto comienza su desplazamiento en la posición de las tres en punto y se va moviendo en sentido contrario a las agujas del reloj.

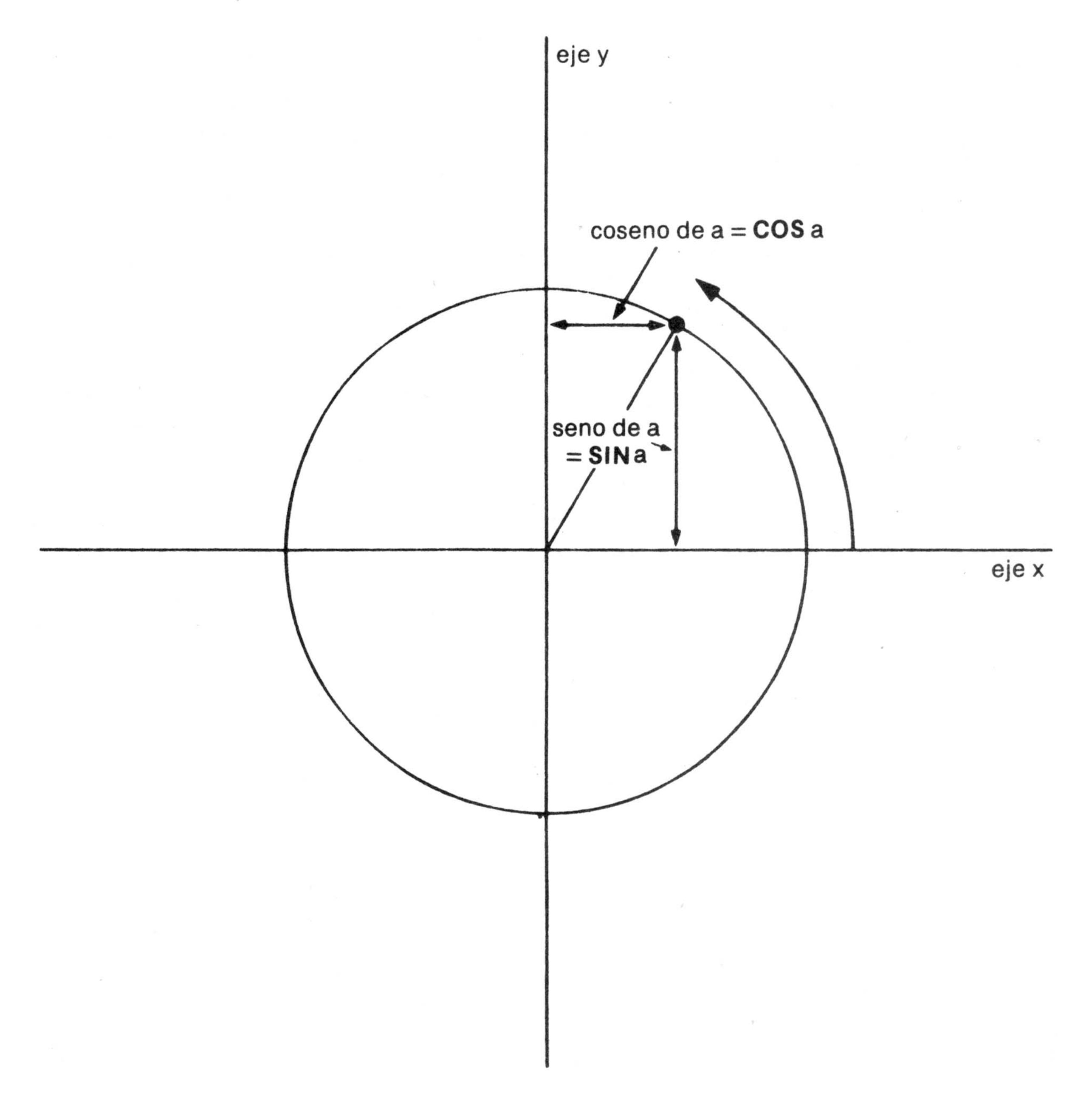


Hemos dibujado también dos líneas denominadas ejes que se cruzan en el centro del círculo. La línea horizontal (entre las posiciones de las tres y de las nueve en punto de un reloj) se denomina el eje *x* y la vertical (entre las posiciones de las 6 *y* de las 12 en punto) se denomina el eje y.

Para especificar dónde está el punto, dirá cuánto se ha desplazado éste alrededor del círculo a partir de su posición de partida de las tres en punto: llamaremos a esta distancia *a*. Sabemos que la longitud de la circunferencia del círculo es 2π (porque su radio es 1 y su diámetro es, pues, 2), por lo que cuando se haya desplazado una cuarta parte del recorrido del círculo, será $a = \pi/2$, cuando se haya desplazado la mitad de su recorrido será $a=\pi$ y cuando haya realizado el recorrido completo, será $a=2\pi$.

Dada la distancia en forma curvilínea alrededor del contorno, *a*, puede considerarse oportuno conocer otras dos distancias para determinar cuánto se ha

desplazado el punto a la derecha del eje **y** y cuánto por encima del eje **x**. Estas se denominan *coseno y seno* de a respectivamente. Las funciones **COS** y **SIN** en el ordenador sirven para calcularlas.



Tenga presente que si el punto se desplaza a la izquierda del eje *y*, entonces, el coseno se hace negativo y si el punto pasa a la región por debajo del eje *x*, el seno se hace negativo.

Otra propiedad es que una vez que a haya alcanzado el valor de 2π, el punto vuelve al punto de partida y el seno y el coseno comienzan a tomar los mismos valores que antes:

SIN (a+2*PI)=SIN a
COS (a+2*PI)=COS a

La tangente de *a* se define como el seno dividido por el coseno y la función correspondiente en el ordenador se denomina **TAN**.

A veces, necesitamos trabajar con estas funciones a la inversa cuando lo que se da es el valor que toman dichas funciones y hemos de determinar el valor de a correspondiente. Las funciones que permiten realizar esta operación se denominan arco seno (**ASN** en el ordenador), arco coseno (**ACS**) y arco tangente (**ATN**).

En el diagrama del punto que se desplaza alrededor del círculo, examine el radio que une el centro con el punto. Debe ser capaz de constatar que la distancia que hemos llamado *a*, la distancia que el punto ha recorrido a lo largo del contorno del círculo, es una forma de medir el ángulo a través del cual el radio se ha desplazado alejándose del eje x. Cuando $a=\pi/2$, el ángulo es de 90°; cuando $a=\pi$ el ángulo es de 180^0 y así sucesivamente, hasta llegar a que sea $a=2\pi$ en que el ángulo será de 360°. Asimismo, podría hacer caso omiso de los grados y medir el ángulo en función de **a** solamente y diremos, entonces, que estamos midiendo el ángulo en *radianes.* Así, $\pi/2$ radianes = 90 grados, etc.

Siempre debe tener presente que, en el ZX Spectrum, las funciones **SIN, COS,** etc., trabajan con radianes y no con grados. Pra la conversión de grados en radianes, hay que dividir por 180 y multiplicar por π; para pasar de radianes a grados, tendrá que dividir por π y multiplicar por 180.

CAPITULO 11

Números aleatorios

Resumen
RANDOMIZE RND

Este capítulo trata de la función **RND** y de la palabra clave **RANDOMIZE.** Se utilizan ambas en relación con los números aleatorios, por lo que debe tener cuidado en no mezclarlas. Las dos están en la misma tecla (**T**), habiéndose tenido que abreviar **RANDOMIZE en RAND.**

En algunas formas, **RND** es como una función: realiza cálculos y da lugar a un resultado. No es habitual por cuanto que no necesita un argumento.

Cada vez que la utiliza, su resultado es un nuevo número aleatorio entre 0 y 1 (a veces, puede tomar el valor 0, pero nunca 1).

Pruebe:

```
10 PRINT RND
20 GO TO 10
```

para ver cómo varía la respuesta. ¿Puede detectar alguna configuración determinada? No será capaz pues «random» significa que es una función aleatoria.

Realmente, **RND** no es verdaderamente aleatoria, porque sigue una secuencia fija de 65536 números. Sin embargo, es tan profundamente complicada la secuencia que no se tienen pautas obvias y por ello diremos que **RND** es psedoaleatoria.

RND da un número aleatorio entre 0 y 1, pero fácilmente puede obtener números aleatorios en otros intervalos. Por ejemplo, **5*RND** está entre 0 y 5 y **1.3+0.7*RND** está entre 1.3 y 2. Para obtener números enteros, utilice **INT** (recordando que **INT** siempre se redondea por defecto) como en **1+INT (RND*6)** que utilizaremos en un programa para simular dados. **RND*6** está en el intervalo de 0 a 6, pero puesto que nunca llega realmente a 6, **INT (RND*6)** es 0, 1, 2, 3, 4 ó 5.

Veamos el programa:

```
10 REM programa de tirada de dados
20 CLS
30 FOR n=1 TO 2
40 PRINT 1+INT (RND*6);" ";
50 NEXT n
60 INPUT A$: GO TO 20
```

Pulse **ENTER** cada vez que quiera tirar los dados.

La sentencia **RADOMIZE** se utiliza para hacer que RND comience en un lugar definido en su secuencia de números, como puede ver con el programa siguiente:

```
10 RANDOMIZE 1
20 FOR n = 1 TO 5: PRINT RND,: NEXT n
30 PRINT : GO TO 10
```

Después de cada ejecución de **RANDOMIZE 1,** la secuencia **RND** vuelve a iniciarse con 0.0022735596. Puede utilizar otros números entre 1 y 65535 en la sentencia **RANDOMIZE** para comenzar la secuencia **RND** en lugares diferentes.

Si tuviera un programa con **RND** en el mismo y también tuviera algunos errores que no hubiera encontrado, entonces sería de utilidad el empleo de **RANDOMIZE** como tal, de modo que el programa se comportara de la misma forma cada vez que lo ejecutare.

RANDOMIZE por sí misma (y **RANDOMIZE 0** tiene el mismo efecto) es diferente, porque realmente hace la función de aleatorizar **RND.** Puede constatarlo en el programa siguiente:

```
10 RANDOMIZE
20 PRINT RND : GO TO 10
```

La secuencia que consigue en este caso no es muy aleatoria, porque **RANDOMIZE** utiliza el tiempo desde que se puso en servicio el ordenador. Puesto que esto se produce en la misma magnitud cada vez que se ejecuta **RANDOMIZE,** la siguiente **RND** hace más o menos lo mismo. Conseguiría un efecto más aleatorio sustituyendo **GO TO 10** por **GO TO 20.**

Observación: La mayoría de las versiones de BASIC utilizan **RND** y **RANDOMIZE** para obtener números aleatorios, pero no todos las utilizan de la misma manera.

Veamos un programa para lanzar monedas y contar los números de caras y cruces.

```
10 LET caras =0: LET cruces =0
20 LET moneda = INT (RND*2)
30 IF moneda =0 THEN LET caras = caras + 1
40 IF moneda = 1 THEN LET cruces =cruces + 1
50 PRINT caras; ",";cruces
60 IF cruces <> 0 THEN PRINT caras/cruces;
70 PRINT : GO TO 20
```

La relación caras/cruces debe llegar a ser aproimadamente 1 si prosigue durante un período de tiempo suficientemente largo, porque, a largo plazo, debe esperar igual número de caras que de cruces.

Ejercicios

1. Pruebe esta regla:
   Supongamos que elige un número entre 1 y 872 y teclee:

   RANDOMIZE su número

   Entonces, el siguiente valor de **RND** será:

   (75* (su número + 1) – 1) / 65536

2. (Para matemáticos solamente)
   Sea *p* un número primo (grande) y sea a una raíz primitiva de módulo p.
   Entonces, si b_i es el resultado de a^i módulo *p* ($1 =< b_i =< p-1$), la secuencia:

Bi – 1
----
p – 1

es una secuencia cíclica de p-1 números distintos en el intervalo 0 a 1 (excluyendo 1). Eligiendo a adecuadamente, estos números pueden aparecer como bastante aleatorios.

65537 es un primo de Fermat, 2^{16}+1. Habida cuenta de que el grupo multiplicativo de resto distinto de no cero del módulo 65537 tiene una potencia de 2 como su orden, un resto es una raíz primitiva si, y solamente si, no es un resto cuadrático. Utilice la ley de Gauss de reciprocidad cuadrática para demostrar que 75 es una raíz primitiva módulo 65537.

El ZX Spectrum utiliza p=65537 y a=75 y almacena algunos valores b_i-*1* en memoria. **RND** obliga la sustitución de b_i-*1* en memoria por b_i + *1- 1 y* proporciona el resultado *(*b_{i+1} -1*), (p-1).* **RANDOMIZE** *n* (con 1 < n < 65535) hace b_i igual a n+1.

RND está casi uniformemente distribuida en el intervalo 0 a 1.

CAPITULO 12

Matrices

Resumen
Matrices (la forma en que el ZX Spectrum trata las matrices de cadenas se aparta algo del procedimiento normal).
DIM ...

Suponga que tiene una lista de números, por ejemplo, las notas de diez personas en una clase. Para almacenarlas en el ordenador podría establecer una variable única para cada persona, pero tal método se consideraría arduo. Podría decidir llamar a la variable Bloggs 1, Bloggs 2 y así sucesivamente hasta Bloggs 10, pero el programa para preparar estos diez números sería bastante largo y aburrido de introducir por teclado.

Sería mucho más atractivo si pudiera teclear:

```
 5 REM este programa no funcionará
10 FOR n = 1 TO 10
20 READ Bloggs n
30 NEXT n
40 DATA 10,2,5,19,16,3,11,1,0,6
```

Bueno, lo cierto es que no puede.

Sin embargo, hay un mecanismo mediante el cual puede aplicar esta idea y utiliza *matrices.* Una matriz es un conjunto de variables, sus elementos, todos ellos con el mismo nombre, y distinguidos solamente por un número (el *subíndice)* escrito entre paréntesis después del nombre. En nuestro ejemplo, el nombre podría ser *b* (como las variables de control de los bucles **FOR-NEXT**, el nombre de una matriz debe ser una sola letra) y las diez variables serían entonces *b*(1), *b(*2*)* y así sucesivamente, hasta *b*(10).

Los elementos de una matriz se denominan variables *con subíndice,* en oposición a las variables *simples,* con las que ya está familiarizado.

Antes de que pueda utilizar una matriz, debe reservar algún espacio para la misma en el interior del ordenador y puede hacer esta operación con el empleo de una sentencia **DIM** (dimensión).

```
DIM b(10)
```

establece una matriz llamada *b* con dimensión 10 (esto es, hay 10 variables con subíndice *b*(1)...*b*(10)) e inicializa los 10 valores a 0. También borra cualquier matriz denominada *b* que existiera anteriormente. (Pero no una variable simple. Una matriz y una variable numérica simple con el mismo nombre pueden coexistir y no debe producirse ninguna confusión entre ellas porque la variable de matriz siempre tiene un subíndice).

El subíndice puede ser una expresión numérica arbitraria, por lo que, ahora, puede escribir:

```
10 FOR n =1 TO 10
20 READ b(n)
30 NEXT n
40 DATA 10,2,5,19,16,3,11,1,0,6
```

También puede establecer matrices con más de una dimensión.

En una matriz de dos dimensiones, necesita dos números para especificar uno de los elementos (bastante similar a los números de línea y de columna que especifican una posición de carácter en la pantalla de televisión), por lo que tiene la forma de una tabla. Como alternativa, si imagina los números de línea y de columna (dos dimensiones) como referencia a una página impresa, podría tener una dimensión adicional para los números de página. Por supuesto, nos estamos refiriendo a matrices numéricas y por lo tanto, los elementos no serían caracteres impresos como en un libro, sino números. Piense en los elementos de una matriz tridimensional **v**, como estando especificados por **v**: número de página, número de línea, número de columna.

Por ejemplo, para establecer una matriz de dos dimensiones **c** con dimensiones 3 y 6, utilizará una sentencia **DIM**:

DIM c(3,6)

Con esta sentencia obtendrá 3*6=18 variables con subíndice.

```
c(1,1),c(1,2)...,c(1,6)
c(2,1),c(2,2)...,c(2,6)
c(3,1),c(3,2)...,c(3,6)
```

El mismo principio se aplica para cualquier número de dimensiones.

Aunque pueda tener un número y una matriz con el mismo nombre, no ocurre así con dos matrices, aun cuando éstas tengan distintos números de dimensiones.

Hay también matrices de cadenas. Las cadenas en una matriz se diferencian de las cadenas simples en que son de longitud fija y en que el modo de asignación es siempre «procusteano» (fragmentado o rellenado con espacios). Otra forma de considerarles es como matrices (con una dimensión adicional) de caracteres simples. El nombre de una matriz de cadena es una letra única seguida por **$** y una matriz de cadena y una variable de cadena simple no pueden tener el mismo nombre (a diferencia con el caso de números).

Suponga, entonces, que quiere una matriz **a$** de cinco cadenas. Debe decidir cuál será la longitud de estas cadenas. Supongamos que fuera suficiente una longitud de 10 caracteres en cada una. Teclee:

DIM a$(5,10)

con lo que obtendrá una matriz de 5*10 de caracteres, pero también puede considerar cada fila como una cadena:

```
a$(1)=a$(1,1) a$(1,2) ... a$(1,10)
a$(2)=a$(2,1) a$(2,2) ... a$(2,10)
   .  .  .          ..           .
   .  .  .          ..           .
a$(5)=a$(5,1) a$(5,2) ... a$(5,10)
```

Si da el mismo número de subíndices (dos, en este caso) como si hubier dimensiones en la sentencia **DIM**; entonces obtendrá un carácter único; pero si omite el último, obtendrá una cadena de longitud fija. Así, por ejemplo, A$(2,7) es el

7.° carácter en la cadena A$(2); conel empleo de la notación de fragmentación, podría escribirlo también como A$(2)(7). Ahora teclee:

```
LET a$(2) = "1234567890"
```

y

```
PRINT A$(2),a$(2,7)
```

Obtendrá:

```
1234567890    7
```

Para el último subíndice (el que puede omitir), también tener una forma de fragmentación, de modo que, por ejemplo, sea:

```
a$(2,4 TO 8) = a$(2)(4 TO 8) = "45678"
```

Recuerde:
En una matriz de cadena, todas las cadenas tienen la misma longitud fija.
La sentencia **DIM** tiene un número adicional (el último) para especificar dicha longitud.
Cuando escriba una variable con subíndice para una matriz de cadena, puede introducir un número adicional, o elemento de fragmentación, para estar en correspondencia con el número adicional en la sentencia **DIM**.
Puede tener matrices de cadenas sin ninguna dimensión. Teclee:

```
DIM a$(10)
```

y encontrará que **a$** se comporta casi como una variable de cadena, con la salvedad de ue siempre tiene la longitud 10 y que la asignación es siempre «procrusteana».

Ejercicios

1. Utilice las sentencias **READ** y **DATA** para establecer una matriz **m$** de doce cadenas en las que **m$(n)** es el nombre del n-ésimo mes. (Sugerencia: La sentencia **DIM** será **DIM m$(12,9)**. Pruébela imprimiendo todas las **m$(n)** (utilice un bucle)).
   Teclee:

```
PRINT "En ";m$(3);" se celebran las Fallas"
```

¿Qué puedo hacer con todos esos espacios?

CAPITULO 13

Condiciones

Resumen
AND, OR
NOT

Vimos en el capítulo 3 cómo una sentencia **IF** adopta la forma

IF condición **THEN**

Las condiciones, entonces, eran las relaciones (= , < , > , <= , >= y <>), que comparan dos números o dos cadenas. También pueden combinar varias de ellas, con el empleo de las operaciones lógicas **AND**, **OR** y **NOT**.

Una relación **AND** y otra relación es verdadera siempre que ambas relaciones sean verdaderas, por lo que podría tener una línea como:

IF a$ = "sí" AND x > 0 THEN PRINT x

en la que **x** solamente se consigue imprimir si **a$** = "sí" y **x** > 0. El BASIC es, en este caso, tan próximo al inglés que parece ser que no vale la pena entrar en detalles. Como en inglés, puede unir muchas relaciones con **AND** y, entonces, el conjunto es verdadero si lo son todas las relaciones individuales.

Una relación **OR(O)** otra relación es verdadera siempre que una de las dos relaciones, por lo menos, sea verdadera (recuerde que sigue siendo verdadera si ambas relaciones son verdaderas; esto es algo que no siempre se implica en inglés).

La relación **NOT** vuelve todo al revés, La relación **NOT** es verdadera siempre que la relación sea falsa y falsa siempre que sea verdadera.

Las expresiones lógicas pueden constituirse con relaciones y las funciones **AND**, **OR** y **NOT**, lo mismo que las expresiones numéricas pueden formarse con números y los signos +, -, etc.; incluso puede ponerles entre paréntesis, en caso de necesidad. Tienen prioridades en la misma forma que las operaciones habituales +, *, / y ↑ , esto es: **OR** tiene la más baja prioridad, luego **AND**, luego **NOT**, luego las relaciones y las operaciones habituales.

NOT es realmente una función, con un argumento y un resultado, pero su prioridad es mucho más baja que la de otras funciones. Por consiguiente, su argumento no precisa paréntesis a no ser que contenga **AND** u **OR** (o ambas funciones). **NOT a = b** significa lo mismo que **NOT (a = b)** (y lo mismo que **a <> b**, por supuesto).

<> es la negación de **=** en el sentido de que es verdadera si, y solamente, si, **=** es falsa. Dicho de otro modo:

a <> b es lo mismo que **NOT a = b**

y también:

NOT a <> b es lo mismo que **a = b**

Persuádase a sí mismo que >= y <= son las negaciones de < y de > respectivamente; por consiguiente, siempre puede prescindir de **NOT** en una relación cambiando la relación.

Además:

NOT (una primera expresión lógica **AND** una segunda)

es lo mismo que

NOT (la primera) **OR NOT** (la segunda)

y

NOT (una primera expresión lógica **OR** una segunda)

es lo mismo que

NOT (la primera) **AND NOT** (la segunda)

Con el empleo de lo anterior, puede efectuar las funciones **NOT** a través de paréntesis hasta que, a la larga, estén aplicadas todas a las relaciones y, luego, puede prescindir de ellas. Hablando en términos lógicos, **NOT** es innecesaria, aunque pueda seguir considerando que su empleo hace más claro un programa.

La siguiente sección es bastante complicada y puede pasarse por alto por los «pusilánimes».

Pruebe:

PRINT 1 = 2,1 <> 2

que podría esperar que diera un error de sintaxis. De hecho, en lo que respecta al ordenador, no hay nada como un valor lógico; en cambio, emplea número ordinarios, con observancia de unas pocas reglas:

(i) = , < , > , <= , =< y <> todos ellos dan resultados numéricos: 1 para verdadero y 0 para falso. Por consiguiente, el comando **PRINT** anterior imprimió 0 para '1 = 2', que es falso y 1 para '1 <> 2' que es verdadero.

(ii) En:

IF condición **THEN ...**

la condición puede ser realmente cualquier expresión numérica. Si su valor es 0, entonces, cuenta como falso y cualquier otro valor (incluyendo el valor de 1 que da una relación verdadera) cuenta como verdadero. Así, la sentencia **IF** significa exactamente lo mismo que:

IF condición <> 0 **THEN ...**

(iii) **AND**, **OR** y **NOT** son también operaciones con valoración numérica.

x AND y tiene el	x, si y es verdadera (no cero) 0 (falsa), si y es falsa (cero)
x OR y tiene el valor	1 (verdadera), si y es verdadera (no cero) x, si y es falsa (cero)
NOT x tiene el valor	0 (falsa), si x es verdadera (no cero) 1 (verdadera), si x es falsa (cero)

(observe que «verdadera» significa «no cero» cuando estamos comprobando un valor determinado, pero significa '1' cuando estamos obteniendo uno nuevo).

Lea detenidamente el capitulo de nuevo teniendo presente lo anteriormente expuesto, cerciorándose de que todo va bien.

En las expresiones **x AND y**, **x OR y** y **NOT x, x** e **y** tomará normalmente los valores 0 y 1 para falso y verdadero. Realice las diez combinaciones diferentes (cuatro para **AND**, cuatro para **OR** *y* dos para **NOT**) *y* compruebe que hacen lo que el capítulo le indica que ha de esperar que hagan.

Pruebe este programa:

```
10 INPUT a
20 INPUT b
30 PRINT(a AND a >=b) + (b AND a<b)
40 GO TO 10
```

Cada vez, imprime el más grande de los dos números **a** y **b**.

Convénzase a sí mismo de que puede acordarse de

x AND y

como teniendo el significado de

x si **y** (si no, el resultado es 0)

y de

x OR y

con el significado de

x a no ser que **y** (en cuyo caso, el resultado es 1)

Una expresión con el empleo de AND o de OR como la anterior se denomina una expresión *condicional.* Un ejemplo con el empleo de **OR** podría ser:

```
LET precio total = precio menos impuesto*(1.15 OR v$ = "
calculado zero")
```

Observe cómo **AND** tiende a ir con la adición (porque su valor por defecto es 0) y **OR** tiende a ir con la multiplicación (porque su valor por defecto es 1).

También puede elaborar expresiones condicionales valoradas pero solamente utilizando AND

X$ AND y tiene el valor { x$ si *y* no es cero
"" si *y* es cero

por lo que significa **x$** si **y** (si no, la cadena vacía).

Pruebe este programa, que introduce dos cadenas y las pone en orden alfabético.

```
10 INPUT "introducir dos cadenas" 'a$,b$
20 I F a$ > b$ THEN LET c$ = a$: LET a$ = b$: LET b$ = c$
30 PRINTa$;" ";("<"AND a$<b$)+("=" AND a$=b$);" ";b$
40 GO TO 10
```

Ejercicio

1. El lenguaje BASIC opera, a veces, de forma distinta al inglés. Considere, por ejemplo, la cláusula inglesa 'si *a* no es igual a *b* o *c*'. ¿Cómo podría escribirla en BASIC? La respuesta no es

   IF A<> B OR C

   ni tampoco

   IF A <> B OR A <>C

CAPITULO

14

El juego de caracteres

Resumen
CODE, CHR$
POKE, PEEK
USR
BIN

Las letras, dígitos, signos de puntuación, etc., que pueden aparecer en cadenas se denominan *caracteres y* constituyen el alfabeto, o *juego de caracteres,* que emplea el ZX Spectrum. La mayoría de estos caracteres son *símbolos* simples, pero hay algunos más, denominados *tokens* (términos simbólicos) que representan a palabras completas, tales como **PRINT, STOP, <>** , etc.

Hay 256 caracteres y cada uno tiene un código entre 0 y 255. Hay una lista completa de ellos en el Apéndice A. Para la conversión entre códigos y caracteres, hay dos funciones **CODE y CHR$.**

CODE se aplica a una cadena y proporciona el código del primer carácter en la cadena (o 0 si la cadena está vacía).

CHR$ se aplica a un número y proporciona la cadena de caracteres única cuyo código es ese número.

Este programa proporciona la impresión del juego completo de caracteres:

```
10 FOR a=32 TO 255: PRINT CHR$ A;: NEXT a
```

En la parte superior puede ver un espacio, 15 símbolos y signos de puntuación, los diez dígitos, siete símbolos más, las mayúsculas, seis símbolos más, las minúsculas y cinco símbolos adicionales. Estos son todos (salvo £ y @) tomados de un juego de caracteres, de gran uso, conocido como ASCII (que es la abreviatura de American Standard Codes for Information Interchange - Códigos normalizados americanos para intercambio de información); ASCII asigna también códigos numéricos a estos caracteres y estos son los códigos que utiliza el ZX Spectrum.

El resto de los caracteres no forman parte del ASCII y son peculiares para el ZX Spectrum. Los primeros entre ellos son un espacio y 15 configuraciones de manchas negras y blancas. Se denominan los símbolos *gráficos* y pueden utilizarse para el dibujo de imágenes. Puede introducirlos a partir del teclado, con el empleo de lo que se llama modo de *gráficos.* Si pulsa **GRAPHICS (CAPS SHIFT** con **9**), entonces, el cursor cambiará a **G**. Ahora, las teclas para los dígitos 1 a 8 darán los símbolos de *gráficos*; por sí mismas, proporcionan los símbolos dibujados en las teclas y con una tecla de cambio pulsada, dan el mismo símbolo pero invertido, esto es, el negro se hace blanco y viceversa.

Prescindiendo de las teclas de cambio, el dígito 9 le lleva de nuevo al modo normal (L) y el dígito 0 es **DELETE.**

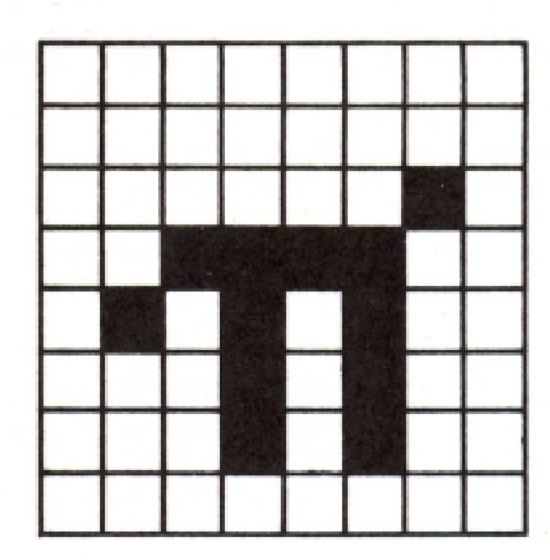
Los dieciséis símbolos gráficos son los siguientes:

Símbolo	Código	Cómo se obtiene	Símbolo	Código	Cómo se obtiene
	128	G 8		143	G cambiada 8
	129	G 1		142	G cambiada 1
	130	G 2		141	G cambiada 2
	131	G 3		140	G cambiada 3
	132	G 4		139	G cambiada 4
	133	G 5		138	G cambiada 5
	134	G 6		137	G cambiada 6
	135	G 7		136	G cambiada 7

Después de los símbolos de gráficos, observará lo que parece ser otra copia del alfabeto de la A a la U. Son caracteres que puede redefinir usted mismo, aunque cuando la máquina se enciende por primera vez constituyen un conjunto de letras. Se denominan gráficos *definidos por el usuario.* Puede introducirlos desde el teclado pasando al modo de gráficos y utilizando luego las teclas de letras de A a U.

Para definir un carácter nuevo, siga esta recomendación: defina un carácter para indicar π

(i) Decida cuál quiere que sea el aspecto del carácter. Cada carácter tiene un cuadrado de puntos de 8 x 8, cada uno de los cuales puede mostrar el color del papel o el color de la tinta (ver el fascículo de introducción). Debe dibujar un diagrama similar al adjunto, con cuadros negros para el color de tinta.



Hemos dejado un margen de un cuadro alrededor de todo el perímetro, porque todas las demás letras tienen dicho margen (salvo las letras minúsculas con rabos, en las que dicho rabo va a la derecha hasta la parte inferior).

(ii) Decida qué gráfico definido por el usuario ha de indicar π. Por ejemplo, la correspondiente a P, de modo que si pulsa P en el modo de gráficos obtendrá π.

(iii) Almacene la nueva configuración. Cada gráfico definido por el usuario tiene su configuarción almacenada como ocho números, uno por cada fila. Puede escribir cada uno de estos números como **BIN** seguida por ocho "0" o "1" (0 para papel, 1 para tinta) de modo que los ocho números para nuestro carácter π son:

BIN 00000000
BIN 00000000
BIN 00000010
BIN 00111100
BIN 01010100
BIN 00010100
BIN 00010100
BIN 00000000

(si tiene conocimientos sobre números binarios, entonces, se dará cuenta de que **BIN** se utiliza para escribir un número en el sistema binario en lugar de hacerlo en el sistema decimal habitual).

Estos ocho números se almacenan en memoria, en ocho lugares, cada uno de los cuales tiene una *dirección.* La dirección del primer byte, o grupo de ocho dígitos, es **USR "P"** (P porque es lo que elegimos en (ii)), la del segundo es **USR "P" + 1** y así sucesivamente, hasta el octavo, que tiene la dirección **USR "P" + 7**.

USR es, en este caso, una función para convertir un argumento de cadena en la dirección del primer byte en memoria para el correspondiente gráfico definido por el usuario. El argumento de cadena debe ser un carácter único que puede ser el gráfico definido por el usuario, o la correspondiente letra (en mayúsculas o en minúsculas). Hay otro uso para **USR**, cuando su argumento es un número, que se tratará más adelante.

Aun cuando no lo comprenda, el siguiente programa lo hará por usted:

```
10 FOR n=0 TO 7
20 INPUT fila: POKE USR "P" + n, fila
30 NEXT n
```

Interrumpirá la introducción de datos (INPUT) ocho veces para permitirle teclear los ocho números **BIN** anteriores. Ha de teclearlos en el orden correcto, comenzando por la fila superior.

La sentencia **POKE** almacena un número directamente en la posición de memoria, eludiendo los mecanismos normalmente utilizados por el BASIC. La opuesta de **POKE** es **PEEK** y nos permite examinar el contenido de una posición de memoria, aunque no altere realmente el contenido de esa posición. Se tratarán adecuadamente en el capitulo 28.

Después de los gráficos definidos por el usuario vienen los 'tokens' (términos simbólicos).

Habrá observado que no hemos impreso los primeros 32 caracteres con códigos 0 a 31. Estos son caracteres de *control.* No producen nada que pueda imprimirse y ya que no tienen efecto sobre la pantalla, o se emplean para controlar algo distinto de ésta se imprime **?** para indicar que no los entiende. Se describen con más detalle en el Apéndice A.

Tres que utiliza la televisión son los correspondientes a los códigos 6, 8 y 13; después de todo **CHR$8** es el único que probablemente encuentre de utilidad.

CHR$ 6 imprime espacios en exactamente la misma forma que una coma en una sentencia **PRINT**, por ejemplo:

```
PRINT 1; CHR$ 6;2
```

hace lo mismo que

```
PRINT 1,2
```

Evidentemente, esta no es una forma muy clara de utilizarlo. Una forma más sutil es:

```
LET a$="1"+CHR$6+"2"
PRINT a$
```

CHR$ 8 es «retroceso»: desplaza la posición de la impresión un espacio hacia atrás. Pruebe:

```
PRINT "1234"; CHR$ 8;"5"
```

que imprime **1235**.

CHR$ 13 es «nueva línea»: desplaza la posición de impresión al principio de la línea siguiente.

En la televisión se utilizan también los correspondientes a los códigos 16 a 23; estos se explican en los capítulo 15 y 16. Todos los caracteres de control se indican en el Apéndice A.

Con el empleo de códigos para los caracteres podemos ampliar el concepto de «ordenación alfabética» para cubrir las cadenas que contengan cualquier tipo de carácter no sólo letras. Si en lugar de pensar sobre la base del alfabeto habitual de 26 letras, utilizamos el alfabeto ampliado de 256 caracteres, en el mismo orden que sus códigos, el principio es exactamente el mismo. Por ejemplo, estas cadenas están en el orden alfabético del ZX Spectrum. (Observe la característica bastante singular de que las letras minúsculas van después de todas las mayúsculas; así, «a» va después de «Z»; también cuentan los espacios)

CHR$ 3 + «JARDINES ZOOLOGICOS»
CHR$ 8+ "CAZA DE ARMADILLOS"
" AAAARGH!"
"(Observación entre paréntesis)”
"100"
"129.95 inc. IVA"
"AASVOGEL" (BUITRE SUDAFRICANO)
"Aardvark" (armadillo)
"PRINT"
"Zoo'
"[Interpolación]"
"aasvogel"
"zoo'
"zoología

He aquí la regla para determinar en qué orden aparecen dos cadenas. Primero, comprar los primeros caracteres. Si son diferentes, entonces, uno de ellos tendrá su código inferior al del otro y la cadena a la cual pertenecen será la anterior (inferior) de las dos cadenas. Si son iguales, entonces, hay que pasar a la comparación de los caracteres siguientes. Si, en este proceso, una de las cadenas se acaba antes que la otra, entonces, esa cadena es la anterior; de cualquier otro modo, deben ser iguales.

Las relaciones =, < , > , <= , >= y <> se utilizan tanto para cadenas como para números: < significa «va antes» y > significa «va después», por lo que

"AA man" < "AARDVARK"
"AARDVARK" > "AA man"

son ambas verdaderas.

<= y >= actúan de la misma forma que lo hacen para números, de modo que:

"La misma cadena" <= "La misma cadena"

es verdadera, pero

"La misma cadena" < "La misma cadena"

es falsa.

Experimente lo anterior con el empleo del programa siguiente, que introduce dos cadenas y las pone en orden.

```
10 INPUT "Teclee dos cadenas:' ,a$,b$
20 IF a$ > b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";
40 IF a$ < b$ THEN PRINT" < "; GO TO 60
50 PRINT "="
60 PRINT " ";b$
70 GO TO 10
```

Observe cómo hemos de introducir **c$** en la línea 20 cuando permutamos **a$** y **b$.**

```
LET a$ = b$: LET b$ = a$
```

no tendría el efecto deseado.

El programa siguiente introduce los gráficos definidos por el usuario para mostrar las piezas de ajedrez:

P para peón
R para torre
N para caballo
B para alfil
K para rey
Q para reina

Piezas de ajedrez.

```
  5 LET b = BIN 01111100: LET c = BIN
    00111000: LET d = BIN 00010000
 10 FOR n = 1 TO 6: READ p$: REM 6 piezas
 20 FOR f =0 TO 7: REM lectura de pieza en 8 bytes
 30 READ a: POKE USR p$+ F,a
 40 NEXT f
 50 NEXT n
100 REM alfil
110 DATA "b",0,d, BIN 00101000, BIN 01000100
120 DATA BIN 01101100,c,b,0
130 REM rey
140 DATA "k",0,d,c,d
150 DATA c, BIN 01000100,c,0
160 REM torre
170 DATA "r",0, BIN 01010100,b,c
180 DATA c,b,b,0
190 REM reina
200 DATA "q",0, BIN 01010100, BIN 00101000,d
210 DATA BIN 01101100,b,b,0
220 REM peón
230 DATA "p",0,0,d,c
240 DATA c,d,b,0
```

```
250 REM caballo
260 DATA "n",0,d,c, BIN 01111000
270 DATA BIN 00011000,c,b,0
```

Observe que 0 puede utilizarse en lugar de BIN 00000000.
Cuando haya ejecutado el programa, examine las piezas pasando al modo de gráficos.

Ejercicios

1. Imagine el espacio para un símbolo dividido en cuatro partes como un pastel. Entonces, si cada cuarta parte puede ser negra o blanca, hay 2 x 2 x 2 x 2 = 16 posibilidades. Encontrarlas todas en el juego de caracteres.

2. Ejecute este programa:

```
10 INPUT a
20 PRINT CHR$ a;
30 GO TO 10
```

Si experimenta con este programa, encontrará que **CHR$ a** está *redondeado* al número entero más próximo y si **a** no está en el intervalo de 0 a 255, entonces, el programa se interrumpe con informe de error **B Integer out of range** (B entero fuera de margen).

3. ¿Cuál de los dos es el inferior?

```
“EVIL”
"evil"
```

4. Indique cómo modificar el programa para establecer los gráficos definidos por el usuario de modo que utilice las sentencias **READ** y **DATA** en lugar de la sentencia **INPUT**.

CAPITULO 15

Más sobre PRINT e INPUT

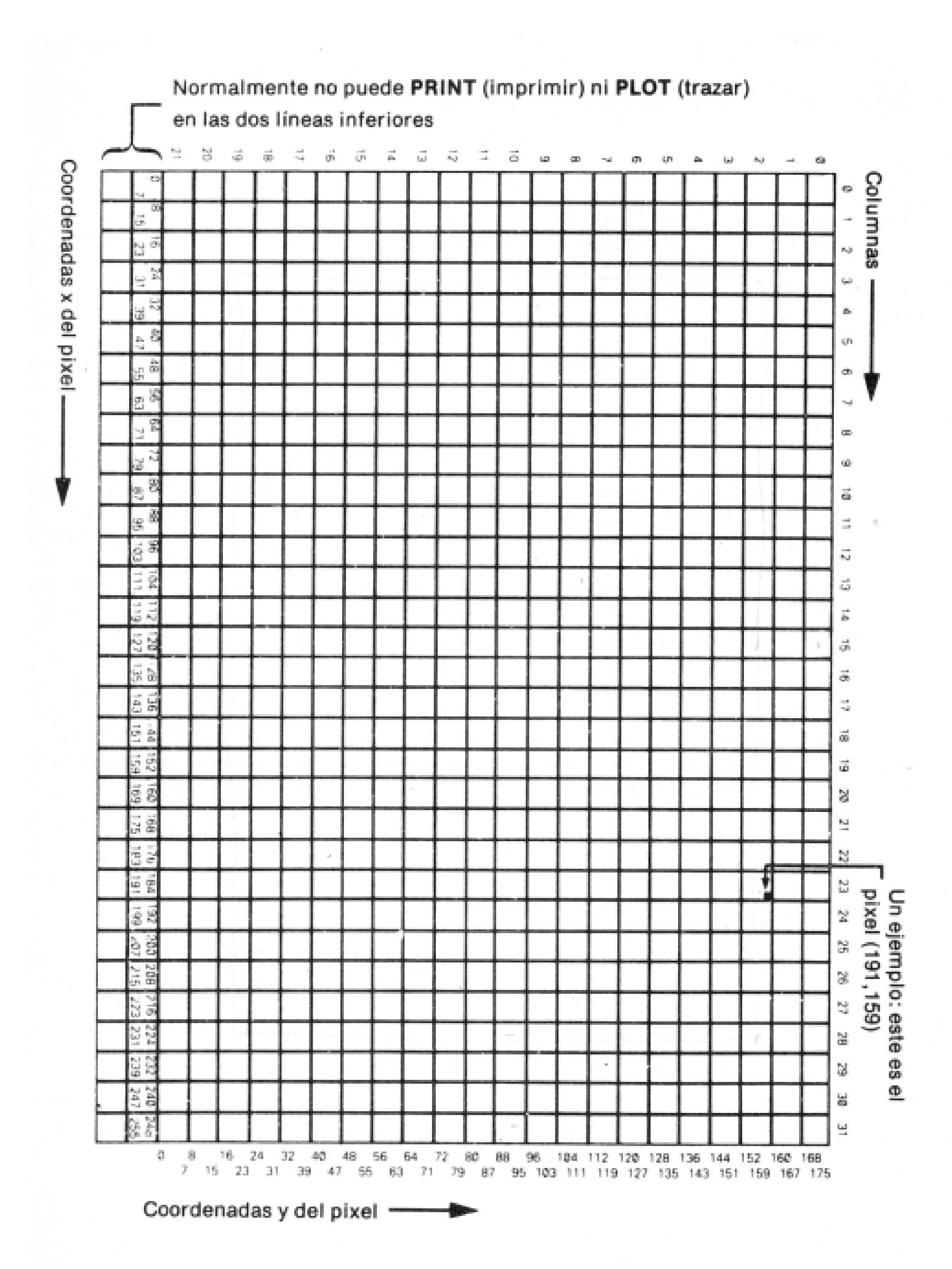
Resumen
CLS
PRINT elementos: nada en absoluto
Expresiones (tipo de cadena o numéricas): **TAB** expresión numérica, **AT** expresión numérica, expresión numérica
PRINT separadores: , ; '
INPUT elementos: variables (numéricas o tipo de cadena)
LINE variable de cadena
Cualquier elemento de impresión **PRINT** que no comience con una letra (los comandos no se consideran como comenzando con una letra).
«Scrolling» (desplazamiento en pantalla).

Ya ha visto **PRINT** utilizado con mucha frecuencia, por lo que tendrá una idea aproximada de cómo se emplea. Las expresiones cuyos valores se imprimen se denominan elementos **PRINT** y están separados por comas o punto y corra. llamados *separadores* **PRINT.** Un elemento **PRINT** puede ser también nada en absoluto, que es una forma de explicar lo que sucede cuando utilice dos comas en una fila.

Hay dos clases más de elementos PRINT, que se emplean para decirle al ordenador que es lo que ha de imprimir sino en donde. Por ejemplo **PRINT AT 11,16; "*"** imprime una estrella en la parte central de la pantalla.

AT línea, columna

desplaza la posición de impresión **PRINT** (o lugar en donde ha de imprimirse el elemento siguiente) a la línea y a la columna especificada. Las líneas están numeradas desde 0 (en la parte superior) a 21 y las columnas desde 0 (a la izquierda) a 31.


Normalmente no puede PRINT (imprimir) ni PLOT (trazar)
en las dos líneas inferiores
Columnas
Coordenadas x del pixel
Un ejemplo: este es el pixel (191,159)
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168
7 15 23 31 39 47 55 63 71 79 87 95 103 111 119 127 135 143 151 159 167 175
Coordenadas y del pixel

TAB columna

imprime suficientes espacios para desplazar la posición de impresión **PRINT** a la columna especificada. Se mantiene en la misma línea o, si ello implicara salto de línea, se desplazaría a la siguiente. Tenga en cuenta que el ordenador reduce el número de columna 'módulo 32' (divide por 32 y se lleva el resto); por ello, **TAB 33** significa lo mismo que **TAB 1**.

Como un ejemplo:

```
PRINT TAB 30;1;TAB 12;"Contenido";AT 3,1;"CAPITULO";
TAB 24;"página"
```

es como podría imprimir el encabezamiento de una página del contenido (o índice) en la página 1 de un libro.

Pruebe la ejecución de este programa:

```
10 FOR n =0 TO 20
20 PRINT TAB 8*n;n;
30 NEXT n
```

Este programa indica lo que se entiende por la reducción de módulo 32 de los números en **TAB**.

Para un ejemplo más elegante, cambie el 8 en la línea 20 por un 6.

Algunas pequeñas puntualizaciones:

(i) Estos nuevos elementos se terminan mejor con punto y coma, como lo hicimos anteriormente.

Puede utilizar comas (o nada, al final de la sentencia), pero ello significa que después de haber establecido cuidadosamente la posición de impresión **PRINT**, cambiará completamente el formato de impresión que había establecido, lo que no suele ser de gran utilidad.

(ii) No puede imprimir en las dos líneas inferiores (22 y 23) de la pantalla porque están reservadas para comandos u órdenes, introducción de datos (**INPUT**), informes, etc. Las referencias a la «línea inferior» suelen significar línea 21.

(iii) Puede utilizar **AT** para poner la posición de impresión incluso en donde ya hay algo impreso; lo antiguo se borrará cuando imprima algo más.

Otra sentencia relacionada con **PRINT** es **CLS**. Esta última borra la pantalla entera, lo que también se realiza por **CLEAR** y **RUN**.

Cuando la impresión llega a la parte inferior de la pantalla, comienza a desplazarse hacia arriba de forma semejante a como se realiza en una máquina de escribir. Puede verlo si teclea:

```
CLS: FOR n = 1 TO 22: PRINT n: NEXT n
```

y luego:

```
PRINT 99
```

unas cuantas veces.

Cuando el ordenador está imprimiendo gran cantidad de información, lo hace cuidadosamente, cerciorándose de que nada se pierde por la parte superior de la pantalla hasta que usted haya tenido oportunidad de examinarlo adecuadamente. Puede constatarlo si teclea:

```
CLS: FOR n = 1 TO 100: PRINT n: NEXT n
```

Cuando haya impreso una pantalla completa, se parará, escribiendo **scroll?** en la parte inferior de la pantalla. Ahora puede inspeccionar los primeros 22 números a su comodidad. Cuando haya acabado su examen, pulse **y** (por «yes») y el ordenador le proporcionará otra pantalla llena de números. Realmente, cualquier tecla hará que el ordenador siga el proceso con la excepción de las teclas **n** (por «no»), **STOP** (**SYMBOL SHIFT** y **a**) o **SPACE** (la tecla **BREAK**). Estas teclas harán que el ordenador detenga la ejecución del programa con un informe **D BREAK-CONT repeats**.

La sentencia **INPUT** puede hacer mucho más de lo que le hemos dicho hasta ahora. Ya ha visto sentencias **INPUT** como

```
INPUT "Qué edad tienes?", edad
```

en la que el ordenador imprime la frase Qué edad tienes? en la parte inferior de la pantalla y luego, tiene que teclear su edad.

De hecho, una sentencia **INPUT** está constituida por elementos y separadores en la misma forma exactamente que lo está una sentencia **PRINT**, por ello **qué edad tienes?** y **edad** son ambos elementos de información de **INPUT**. Estos últimos suelen ser los mismos que los de **PRINT**, pero hay algunas diferencias muy importantes.

En primer lugar, un elemento adicional de **INPUT** es la variable cuyo valor ha de introducir por teclado (**edad** en nuestro anterior ejemplo). La regla es que si un elemento de **INPUT** comienza con una letra, debe ser una variable cuyo valor ha de introducirse.

En segundo lugar, esto parecería significar que no puede imprimir los valores de variables como parte de un epígrafe; sin embargo, puede eludirlo poniendo paréntesis encerrando a la variable. Cualquier expresión que comience con una letra debe encerrarse entre paréntesis si ha de imprimirse como parte de un epígrafe.

Cualquier clase de elemento de impresión **PRINT** que no sea afectado por estas reglas es también un elemento de introducción **INPUT**.

Veamos un ejemplo que sirva de ilustración de lo que se está exponiendo:

```
LET mi edad = INT (RND*100): INPUT("Tengo"; mi edad;".");
"Qué edad tienes?", tu edad
```

mi edad está entre paréntesis, por lo que su valor consigue ser impreso. **Tu edad** no está entre paréntesis, por lo que ha de introducir por el teclado su valor.

Todo lo que una sentencia **INPUT** escribe sale en la parte inferior de la pantalla, que actúa con cierta independencia de la mitad superior. En particular, sus líneas están numeradas con respecto a la línea superior de la mitad inferior, aun cuando esta última se haya desplazado hacia arriba de la pantalla de televisión real (lo que sucede si introduce grandes cantidades de datos por **INPUT**).

Para ver cómo opera **AT** en las sentencias **INPUT**, trate de ejecutar la siguiente:

```
10 INPUT "Esta es la línea 1.",a$; AT 0,0;"Esta es la línea 0.",
a$; AT 2,0;"Esta es la línea 2.",a$; AT 1,0;"Esta es todavía
la línea 1.",a$
```

(basta pulsar **ENTER** cada vez que se detenga). Cuando **Esta es la línea 2** es objeto de impresión, la parte inferior de la pantalla se desplaza hacia arriba para dejar

espacio para ella; pero la numeración se desplaza también hacia arriba, de modo que las líneas de texto mantienen sus mismos números.

Ahora, pruebe el programa siguiente:

10 FOR n =0 TO 19: PRINT AT n,0;n;: NEXT n
20 INPUT AT 0,0;a$; AT 1,0; a$: AT 2,0;a$; AT 3,0;a$; AT 4,0;a$; AT 5,0;a$;

A medida que la parte inferior de la pantalla se va desplazando cada vez más hacia arriba, la parte superior no se altera hasta que la parte inferior «amenace» con escribir en la misma línea que la posición de impresión **PRINT**. Entonces, la parte superior comienza a desaparecer por arriba de la pantalla (acción de «scrolling») para evitar que dicha amenaza se «haga realidad».

Otro perfeccionamiento de la sentencia **INPUT** que no hemos visto todavía se llama introducción **LINE** y es una forma distinta de introducir variables de entrada. Si escribe **LINE** antes del nombre de una variable de entrada que ha de introducirse, como en

INPUT LINE a$

entonces, el ordenador no le proporcionará las comillas de cadena que es lo que suele hacer para una variable de cadena, aunque hará como si estuvieran. Por consiguiente, si teclea

cat

como dato de **INPUT**, a **a$** se le dará el valor **cat**. Puesto que las comillas de cadena no aparecen en la cadena, usted no puede suprimirlas ni introducirlas por teclado en una clase distinta de expresión de cadena para los datos de **INPUT**. Recuerde que no puede utilizar **LINE** para variables numéricas.

Los caracteres de control **CHR$ 22** y **CHR$ 23** tienen efectos bastante similares a **AT** y **TAB**. Son bastante singulares como caracteres de control porque siempre que se envían para ser impresos en la pantalla de televisión, deben ir seguidos por dos caracteres más que no tienen su efecto habitual: se tratan como números (sus códigos) para especificar la línea y la columna (por **AT**) o la posición de tabulación (por **TAB**). Casi siempre encontrará más fácil utilizar **AT** y **TAB**, en la forma habitual, en lugar de los caracteres de control, pero podrían ser de utilidad en algunas circunstancias. El carácter de control **AT** es **CHR$ 22**. El primer carácter después del mismo especifica el número de línea y el segundo el número de columna, de modo que

PRINT CHR$ 22+ CHR$1 + CHR$ c;

tiene exactamente el mismo efecto que

PRINT AT 1,c;

Esto es así aun cuando **CHR$ 1** o **CHR$ c** tuvieran normalmente un significado diferente (por ejemplo, si **c=13**); el carácter **CHR$ 22** antes de ellos anula ese significado.

El carácter de control de **TAB** es **CHR$ 23** y los dos caracteres después del mismo se utilizan para dar un número entre 0 y 65535 que especifica el número que hubiera tenido en un elemento de información de **TAB**:

PRINT CHR$ 23 + CHR$ a + CHR$ b;

tiene el mismo efecto que

PRINT TAB a + 256*b;

Puede emplear **POKE** para detener el ordenador, que le hará la pregunta **scroll?**, escribiendo

POKE 23692,255

periódicamente. Después de esto, se desplazará hacia arriba en la pantalla 255 veces antes de pararse con la pregunta **scroll?** Como un ejemplo, pruebe:

```
10 FOR n =0 TO 10000
20 PRINT n: POKE 23692,255
30 NEXT n
```

y vigile cualquier zumbido (!) de la pantalla, por el movimiento con gran rapidez.

Ejercicios

1. Pruebe el siguiente programa con algunos niños para probar sus tablas de multiplicación.

```
 10 LET m$= ""
 20 LET a = INT (RND*12)+ 1: LET b = INT (RND*12)+ 1
 30 INPUT (m$) ' ' " cuantas son ";(a);”*";(b);”?”;c
100 IF c =a*b THEN LET m$ = "Correcto.": GO TO 20
110 LET m$ = "Erróneo. prueba de nuevo.": GO TO 30
```

Si son hábiles, podrían ingeniárselas para no tener que hacer los cálculos por sí mismos. Por ejemplo, si el ordenador les pide que escriban la respuesta a 2*3, todo lo que tienen que teclear es 2*3.

Una forma de evitarlo es hacerles introducir cadenas en lugar de números. Sustituir **c** en la línea **30** por **c$** y en la línea **100** por **VAL c$** e insertar una línea.

```
40 IF c$ <> STR$ VAL c$ THEN LET m$= "Tecleelo
    adecuadamente, como un número.":GO TO
```

Esto los volverá locos. Transcurridos unos días, sin embargo, uno de ellos quizás descubra que pueden eludirlo borrando las comillas de cadena y tecleando **STR$ (2*3)**. Para «cerrar esta escapatoria», puede sustituir **c$** en la línea 30 por **LINE c$**.

CAPITULO 16

Colores

Resumen
INK, PAPER, FLASH, BRIGHT, INVERSE, OVER
BORDER

Ejecute este programa:

```
 10 FOR m =0 TO 1: BRIGHT m
 20 FOR n = 1 TO 10
 30 FOR c=0 TO 7
 40 PAPER c: PRINT "    ";: REM 4 espacios coloreados
 50 NEXT c: NEXT n: NEXT m
 60 FOR m =0 TO 1: BRIGHT m: PAPER 7
 70 FOR c=0 TO 3
 80 INK c: PRINT c;"    ";
 90 NEXT c: PAPER 0
100 FOR c=4 TO 7
110 INK c: PRINT c;"    ";
120 NEXT c: NEXT m
130 PAPER 7: INK 0: BRIGHT 0
```

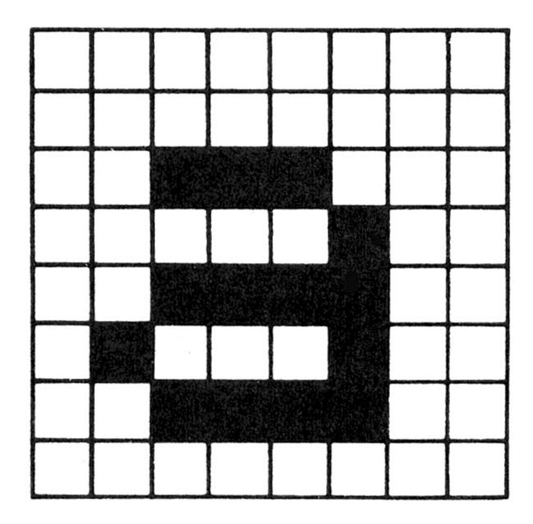
Muestra los ocho colores (incluyendo blanco y negro) y los dos niveles de brillo que el ZX Spectrum puede producir en un televisor en color (si su televisión es de blanco y negro, verá diversas tonalidades de grises). Damos a continuación una lista de colores para referencia; se escriben también con las teclas de números adecuadas.

0 - negro
1 - azul
2 - rojo
3 - púrpura o magenta
4 - verde
5 - azul claro, técnicamente llamado «cyan»
6 - amarillo
7 - blanco

En una televisión de blanco y negro, estos números están en orden de brillo o luminosidad.

Para utilizar adecuadamente estos colores, necesita conocer un poco sobre cómo se dispone la imagen.

La imagen está dividida en 768 (24 líneas de 32) posiciones en donde



pueden imprimirse caracteres y cada carácter se imprime como un cuadrado de 8x8 puntos como para **a**. Esto debe recordarlo de cuando se trató de los gráficos definidos por el usuario en el capítulo 16, en donde teníamos «ceros» para los puntos blancos y «unos» para los puntos negros.

La posición de carácter se ha asociado también con dos colores: la *tinta*, o color de primer plano, que es el color para los puntos negros en nuestro cuadrado y el *papel*, o color de fondo, que se utiliza para los puntos blancos. Para comenzar, cada posición tiene tinta negra y papel blanco, por lo que la escritura aparece como negro sobre blanco.

La posición de carácter tiene también un brillo (brillo normal o especial) y una característica de parpadeo o sin parpadeo (este parpadeo se consigue basculando los colores de tinta y de papale). Todo ello se puede codificar en números, de modo que una posición de carácter tiene:

(i) un cuadrado de 8x8 de «ceros» y de «unos» para definir la forma del carácter, con 0 para papel y «1» para tinta,

(ii) colores de tinta y de papel, cada uno codificado en un número entre 0 y 7,

(iii) un brillo - 0 para brillo normal, 1 para brillo especial y

(iv) un número parpadeante - 0 para fijo, 1 para parpadeo.

Tenga presente que puesto que los colores de tinta y de papel cubren una posición de carácter completa , posiblemente no pueda tener más de dos colores en un bloque dado de 64 puntos . Lo mismo es válido para el número de parpadeo y brillo; se refieren a la posición de carácter completa y no a puntos individuales. Los colores, brillo y número de parpadeo en una posición dada se denominan *atributos*.

Cuando imprime algo en la pantalla, cambia la configuración de puntos en esa posicion; es menos obvio, pero no obstante cierto, que cambia también los atributos en esa posición. Al comenzar no lo observará porque todo se imprime con tinta negra sobre papel blanco (y brillo normal y sin parpadeo), pero puede variarlo con las sentencias **INK**, **PAPER**, **BRIGHT** y **FLASH**. Pruebe

PAPER 5

y luego, imprima algunas cosas. Todo aparecerá sobre papel «cyan» (azul-verdoso pálido), porque a medida que se imprimen, los colores del papel, en las posiciones que ocupan, se ponen a «cyan» (que tiene código 5).

Las demás operan de la misma forma, por lo que después de

PAPER número entre 0 y 7
INK número entre 0 y 7
BRIGHT 0 ó 1
o } considere 0 como desactivado y 1 como activado
FLASH 0 ó 1

cualquier impresión establecerá el atributo correspondiente en todas las posiciones de carácter que utilice. Pruébelo con algunos ejemplos. Ahora debe ser capaz de ver cómo operaba el programa al principio (recuerde que un espcaio es un carácter que tiene el mismo color de tinta (INK) y de papel (PAPER)).

Hay algunos números más que puede utilizar en estas sentencias que tienen efectos menos directos.

El número 8 puede utilizarse en las cuatro sentencias y significa «transparente» en el sentido de que el anterior atributo se transparenta. Supongamos, por ejemplo, que introduce

PAPER 8

entonces, ninguna posición de carácter tendrá su color de papel puesto a 8, porque no existe ningún color correspondiente; lo que sucede es que cuando se imprime una posición, su color de papel se deja el mismo que había antes. **INK 8**, **BRIGHT 8** y **FLASH 8** operan de la misma forma para los demás atributos.

El número 9 sólo puede utilizarse con **PAPER** e **INK** y significa «contraste». El color (tinta o papel) que utiliza con uno de ellos ha de estar en contraste con el del otro, de modo que sea blanco si el otro es oscuro (negro, azul, rojo o magenta) y negro si el otro es claro (verde, cyan, amarilo o blanco).

Pruebe lo anterior tecleando:

INK 9: FOR c =0 TO 7: PAPER c: PRINT c: NEXT c

Una visualización más destacada de su potencia ha de ejecutar el programa al principio para obtener bandas coloreadas y luego introduciendo:

INK 9: PAPER 8: PRINT AT 0,0;: FOR n=1 TO 1000: PRINT n;: NEXT n

En este caso, el color de la tinta siempre estará en contraste con el color del papel anterior en cada posición.

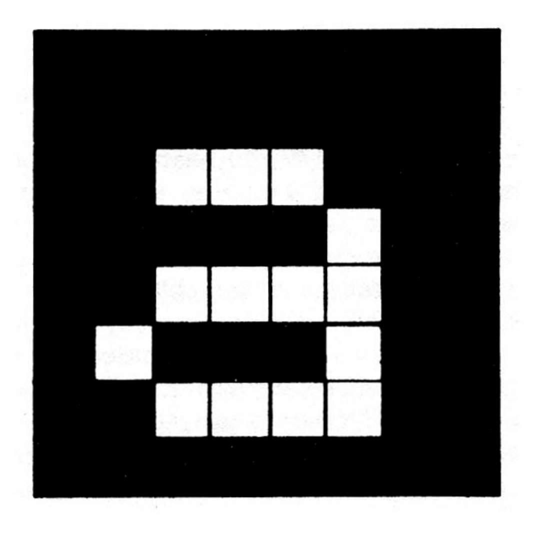
La televisión en color se basa en el hecho bastante curioso de que el ojo humano sólo puede ver realmente tres colores, que son los primarios: azul, rojo y verde. Los demás colores son mezclas de ellos. Por ejemplo, el magenta se consigue mezclando azul con rojo (que es la razón de que su código 3 sea la suma de los códigos correspondientes a los colores azul y rojo).

Para ver cómo los ocho colores se acoplan juntos, imagine tres proyectores rectangulares, con colores azul, rojo y verde, que inciden en un lugar no coincidente en un recinto de papel blanco en la oscuridad En donde se solapen verá mezclas de colores, como se indica por el siguiente programa (observe que los espacios de tinta se obtienen con el empleo de una u otra tecla **SHIFT** con **8**, cuando se está en el modo G).

```
  10 BORDER 0: PAPER 0: INK 7: CLS
  20 FOR a = 1 TO 6
  30 PRINT TAB 6; INK 2;"██████████████████" :REM 18 cuadrados
     de tinta
  40 NEXT a
  50 LET dataline = 200
  60 GO SUB 1000
  70 LET dataline = 210
  80 GOSUB 1000
  90 STOP
 200 DATA 2,3,7,5,4
 210 DATA 2,2,6,4,4
1000 FOR a = 1 TO 6
1010 RESTORE dataline
1020 FOR b= 1 TO 5
1030 READ c: PRINT INK c;"██████";: REM 6 cuadrados de tinta
1040 NEXT b: PRINT : NEXT a
1050 RETURN
```

Hay una función denominada **ATTR** que averigua cuáles son los atributos que están en una posición dada en la pantalla. Se trata de una función bastante complicada, por lo que se ha relegado al final de este capítulo.

Hay otras dos sentencias, **INVERSE Y OVER**, que no controlan los atributos, sino las configuraciones de puntos que se imprimen en la pantalla. Utilizan los números 0 para desactivación y 1 para activación, en la misma forma que para el parpadeo (**FLASH**) o para el brillo (**BRIGHT**), pero ellas no son las únicas posibilidades. Si hace **INVERSE 1**, entonces, las configuraciones de puntos impresas serán las inversas de su forma habitual: los puntos de papel serán sustituidos por puntos de tinta y viceversa. Así, **a** se imprimirla como



Si (como en la condición de encendido) tenemos tinta negra y papel blanco, entonces, esta **a** aparecerá como blanco sobre negro (pero seguimos teniendo tinta negra y papel blanco en esa posición de carácter). Son los puntos los que han cambiado.

La sentencia

OVER 1

pone en acción una clase especial de sobreimpresión. En condiciones normales, cuando algo se escribe en una posición de carácter, borra completamente lo que había antes en la misma; pero, ahora, el carácter nuevo simplemente se añadirá en la parte superior del anterior (véase Ejercicio 3). Ello puede ser especialmente útil para escribir caracteres compuestos, como letras con acentos, como es el caso del siguiente programa para imprimir letras alemanas góticas (una «o» con una diéresis sobre ella. (Haga **NEW** primero).

```
10 OVER 1
20 FOR n = 1 TO 32
30 PRINT "o"; CHR$ 8; """";
40 NEXT n
```

(observe el carácter de control **CHR$ 8** que hace retroceder un espacio).

Hay otra forma de utilizar **INK**, **PAPER**, etc., que probablemente encontrará de más utilidad que como sentencias. Puede ponerles como elementos de información en una sentencia **PRINT** (seguidos por ;) y, entonces, hacen exactamente lo mismo que si se hubieran empleado como sentencias, con la salvedad de que su efecto sólo es temporal: duran hasta el final de la sentencia **PRINT** que las contiene. Así, si teclea.

PRINT PAPER 6;"x";: PRINT "y"

entonces, solamente la x estará en amarillo.

INK y el resto. cuando se utilizan como sentencias, no afectan a los colores de la parte inferior de la pantalla, en donde se introducen los comandos y los datos de **INPUT**. La parte más baja de la pantalla utiliza el color del contorno para su color de papel y el código 9 de contraste para su color de tinta tiene desactivado el parpadeo y todo con brillo normal. Puede cambiar el color del contorno a cualquiera de los ocho colores normales (excepto 8 ó 9) con el empleo de la sentencia

BORDER color

Cuando hace una entrada de datos, se sigue esta norma de empleo de tinta de contraste en papel coloreado en el contorno; pero puede cambiar el color de los epígrafes escritos por el ordenador utilizando los elementos de INK y PAPER (y así sucesivamente) en la sentencia INPUT, lo mismo que lo haría en una sentencia PRINT. Su efecto dura bien hasta el final de la sentencia o bien hasta que se tecleen algunos datos en **INPUT**, lo que suceda primero. Pruebe:

INPUT FLASH 1; INK 1; "Cuál es su número?";n

Hay otra forma de cambiar los colores con el empleo de caracteres de control (de manera bastante similar a la utilizada con los caracteres de control para **AT** y **TAB** en el capitulo 15.

CHR$ 16 corresponde a **INK**
CHR$ 17 corresponde a **PAPER**
CHR$ 18 corresponde a **FLASH**
CHR$ 19 corresponde a **BRIGHT**
CHR$ 20 corresponde a **INVERSE**
CHR$ 21 corresponde a **OVER**

Cada uno de ellos va seguido por un carácter que indica un color mediante su código; así, por ejemplo:

PRINT CHR$ 16+CHR$9;...

tiene el mismo efecto que:

PRINT INK 9; ...

De todos modos, no se habría molestado en utilizar estos caracteres de control porque habría preferido emplear los elementos de color. Sin embargo, algo de gran utilidad que puede hacer con ellos es ponerles en programas, de modo que las partes diferentes que se incluyen sean objeto de listado en colores distintos, para separarlos entre si o incluso para darles un aspecto más atractivo. Debe ponerlos después del número de línea o se perderán.

Para incluirlos en el programa, ha de introducirlos por el teclado, principalmente utilizando el modo extendido con los dígitos.

Los dígitos 0 a 7 establecen el color correspondiente (de tinta si se pulsa también **CAPS SHIFT**, de papel si no se pulsa).

Más precisamente, si está en el modo E y pulsa un dígito (por ejemplo, 6 para amarillo; de cualquier modo, ha de estar entre 0 y 7, nunca 8 ó 9), entonces, se insertan dos caracteres: primero, **CHR$ 17** para papel y luego, **CHR$ 6** que significa «ponerlo en amarillo». Si hubiera estado pulsando **CAPS SHIFT** cuando pulsó el dígito, hubiera conseguido **CHR$ 16** que significa «poner color de tinta» en lugar de **CHR$ 17**.

Puesto que estos son dos caracteres con los que puede obtener algunos efectos singulares cuando los borre, debe pulsar **DELETE** dos veces. Después de la primera pulsación conseguirá, con frecuencia, un signo de interrogación o aparecerán cosas todavía más singulares. No se preocupe, le bastará con pulsar DELETE otra vez.

🡐 y 🡒 pueden comportarse también extrañamente mientras el cursor se esté desplazando más allá de los caracteres de control. Todavía en el modo extendido:

8 da **CHR$ 19** y **CHR$ 0** para brillo normal
9 da **CHR$ 12** y **CHR$ 1** para brillo especial
CAPS SHIFT con 8 da **CHR$ 18** y **CHR$ 0** para no parpadeo
CAPS SHIFT con 9 da **CHR$ 19** y **CHR$ 1** para parpadeo

Hay un par más en el modo ordinario (L):

CAPS SHIFT con 3 da **CHR$ 20** y **CHR$ 0** para caracteres normales
CAPS SHIFT con 4 da **CHR$ 20** y **CHR$ 1** para caracteres inversos

Para resumir, damos a continuación una descripción completa de la fila superior en el teclado:

MODO	CAMBIO	1	2	3	4	5	6	7	8	9	Ø
E	SYMBOL	**DEF FN**	**FN**	**LINE**	**OPEN #**	**CLOSE #**	**MOVE**	**ERASE**	**POINT**	**CAT**	**FORMAT**
	CAPS	Tinta azul	Tinta roja	Tinta magenta	Tinta verde	Tinta «cyan»	Tinta amarilla	Tinta blanca	Sin parpadeo	Con parpadeo	Tinta negra
	NADA	Papel azul	Papel rojo	Papel magenta	Papel verde	Papel «cyan»	Papel amarillo	Papel blanco	Brillo normal	Brillo especial	Papel negro
G	**CUAL-QUIERA**									SALIDA GRAFICOS	**DELETE**
	NADA									SALIDA GRAFICOS	**DELETE**
K,L o C	**CAPS**	**EDIT**	CAPS LOCK	TRUE VIDEO	INVERSE VIDEO	⇦	⇩	⇧	⇨	MODO GRAFICOS	**DELETE**
	SYMBOL	!	@	#	$	%	&	'	(	)	_
	NADA	1	2	3	4	5	6	7	8	9	Ø

La función **ATTR** tiene la forma

ATTR (línea, columna)

Sus dos argumentos son los números de línea y de columna que utilizaría en un elemento AT y su resultado es un número que indica los colores y así sucesivamente en la posición de carácter correspondiente en la pantalla de televisión. Puede utilizarlo tan libremente en expresiones como pueda hacerlo con cualquier otra función.

El número resultante es la suma de otros cuatro números como sigue:
128 si la posición de carácter está parpadeando, 0 si no parpadea
64 si la posición de carácter es brillante, 0 si es normal
8* código para el color del papel
número correspondiente al color de la tinta.

Por ejemplo, si la posición de carácter está parpadeando y tiene brillo normal con papel amarillo y tinta azul, entonces, los cuatro números que hemos de sumar son 128, 0, 8*6 (=48) y 1, lo que hace un total de 177. Pruebe lo anterior con:

```
PRINT AT 0,0; FLASH 1; PAPER 6; INK 1; " "; ATTR (0,0)
```

Ejercicios

1. Pruebe

```
PRINT "B"; CHR$ 8; OVER 1;”/”
```

en donde la barra / atraviesa la B y ha dejado un punto blanco. Esta es la forma en que la sobreimpresión actúa en el ZX Spectrum: dos papeles o dos tintas dan un papel, uno de cada uno da una tinta. Ello tiene la propiedad interesante de que si sobreimprime con la misma cosa dos veces volverá a lo que tenía al principio. Si, ahora teclea:

```
PRINT CHR$ 8; OVER 1;"/"
```

¿por qué recupera una B impoluta sin barra?

2. Teclee:

```
PAPER 0: INK 0
```

¿no es lo mismo que estas no afecten a la parte inferior de la pantalla? Ahora, teclee:

```
BORDER 0
```

y vea lo bien que le trata el ordenador (!)

3. Ejecute este programa

```
10 POKE 22527+ RND*704, RND*127
20 GO TO 10
```

No importa cómo opera; está cambiando los colores de los cuadrados en la pantalla de televisión y las RND deben conseguir que ello suceda aleatoriamente. Las bandas diagonales que quizás vea son una manifestación de la configuración oculta en RND (la configuración que le hace pseudoaleatoria en lugar de realmente aleatoria.

4. Teclee, o haga la función de carga **LOAD** de los caracteres de piezas de ajedrez del capítulo 14 y luego introduzca por el teclado el siguiente programa que dibuja un diagrama de una posición de ajedrez que las emplea.

```
  5 REM dibujo del tablero sin fichas
 10 LET bb = 1: LET bw = 2: REM rojo y azul para tablero
 15 PAPER bw: INK bb: CLS
 20 PLOT 79,128: REM contorno
 30 DRAW 65,0: DRAW 0,-65
 40 DRAW -65,0: DRAW 0,65
 50 PAPER bb
 60 REM tablero
 70 FOR n=0 TO 3: FOR m=0 TO 3
 80 PRINT AT 6+2*n, 11 +2*m;" "
 90 PRINT AT 7+2*n, 10 +2*m;" "
100 NEXT m: NEXT n
110 PAPER 8
120 LET pw=6: LET pb=5: REM colores de piezas blancas y negras
200 DIM b$(8,8): REM posiciones de piezas
205 REM preparación de las posiciones iniciales
210 LET b$(1)= "rnbqkbnr"
220 LET b$(2) = "pppppppp"
230 LET b$(7) = "PPPPPPPP"
240 LET b$(8)= "RNBQKBNR"
300 REM visualización del tablero
310 FOR n=1 TO 8: FOR m=1 TO 8
320 LET bc=CODE b$(n,m); INK pw
325 IF bc = CODE" " THEN GO TO 350: REM espacio
330 IF bc > CODE "Z" THEN INK pb: LET bc=bc-32: REM
      minúsculas para negro
340 LET bc = bc + 79: REM conversión a gráficos
350 PRINT AT 5+n, 9+m; CHR$ bc
360 NEXT m: NEXT n
400 PAPER 7: INK 0
```

CAPITULO 17

Gráficos

Resumen
PLOT, DRAW, CIRCLE
POINT
pixels

En este capítulo veremos cómo dibujar imágenes en el ZX Spectrum. La parte de la pantalla que puede utilizar tiene 22 líneas y 32 columnas, lo que equivale a 22*32=704 posiciones de carácter. Como quizás recuerde del capítulo 16, cada una de estas posiciones de crácter está constituida por cuadrados de 8 x 8 puntos y estos se denominan «pixels» (elementos de imagen).

Un pixel está especificado por dos números, que son sus *coordenadas*. La primera, su *coordenada x*, indica la distancia en sentido transversal desde la columna izquierda extrema (recuerde que x es una cruz, en el sentido transversal); la segunda, su *coordenada y*, indica la altura desde la parte inferior. Estas coordenadas suelen escribirse como un par de números entre paréntesis, por lo que (0,0), (255,0), (0,175) y (255,175) son las esquinas inferior izquierda, inferior derecha, superior izquierda y superior derecha, respectivamente.

La sentencia

PLOT coordenada x, coordenada y

entinta el pixel con esta coordenadas, de modo que el siguiente programa «sarampión»

```
10 PLOT INT (RND*256), INT (RND*176): INPUT a$: GO TO 10
```

dibuja un punto aleatorio cada vez que pulsa **ENTER**.

Veamos, ahora, un programa bastante más interesante. Traza una representación gráfica de la función **SIN** (una onda sinusoidal) para los valores entre 0 y 2π.

```
10 FOR n =0 TO 255
20 PLOT n,88+80*SIN (n/128*PI)
30 NEXT n
```

El siguiente programa traza un gráfico de **SQR** (parte de una parábola) entre 0 y 4:

```
10 FOR n =0 TO 255
20 PLOT n, 80*SQR(n/64)
30 NEXT n
```

Observe que las coordenadas del pixel son bastante diferentes de la línea y columna en un **AT**. Puede encontrar el diagrama en el capítulo 15 de utilidad cuando trabaje con las coordenadas del pixel y los números de línea y de columna.

Para ayudarle en sus dibujos, el ordenador trazará líneas rectas, círculos y arcos, con el empleo de las sentencias **DRAW** y **CIRCLE**.

La sentencia **DRAW**, para dibujar una línea recta, adopta la forma

DRAW x,y

El lugar de partida de la línea es el pixel en donde quedó la última sentencia **PLOT**, **DRAW** o **CIRCLE** (se denomina posición **PLOT**; **RUN**, **CLEAR**, **CLS** y **NEW** le repone a la esquina inferior izquierda, en (0,0)) y el lugar de final del trazado es **x** pixels a la derecha del mismo e **y** pixels arriba. La sentencia **DRAW**, en sí misma, determina la longitud y la dirección de la línea, pero no su punto de partida.

Observe que los números en una sentencia **DRAW** pueden ser negativos, aunque los incluidos en una sentencia **PLOT** no lo pueden ser.

Asimismo, puede trazar y dibujar en color, aunque ha de tener presente que los colores cubren siempre la totalidad de una posición de carácter y no puede especificarse para pixels individuales. Cuando se traza un pixel, se ajusta para mostrar el color de tinta en plenitud y a la totalidad de la posición de carácter que le contiene se le da el color de tinta corriente. El siguiente programa lo pone de manifiesto:

```
10 BORDER 0: PAPER 0: INK 7: CLS : REM apagar pantalla
20 LET x1=0: LET y1=0: REM comienzo de línea
30 LET c=1: REM para color de tinta, comienzo azul
40 LET x2 = INT (RND*256): LET y2 = INT (RND*176): REM
   acabado aleatorio de línea
50 DRAW INK c;x2-x1, y2-y1
60 LET x1 = x2: LET y1 = y2: REM siguiente línea comienza en
    donde acabó la última
70 LETc=c+1: IF c=8 THEN LETc=1: REM nuevo color
80 GO TO 40
```

Las líneas parecen ser más anchas a medida que prosigue el programa y ello se debe a que una línea cambia los colores de todo el entintado en pixels de todas las posiciones de carácter que atraviesa. Observe que puede encajar los elementos **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE** y **OVER** en una sentencia **PLOT** o **DRAW**, lo mismo que podría hacerlo con PRINT e INPUT. Van entre la palabra clave y las coordenadas y están terminadas por punto y coma o comas.

Una característica adicional de **DRAW** es que puede utilizarlo para dibujar arcos de circunferencia en lugar de líneas rectas, con el empleo de un número suplementario para especificar un ángulo a girar; la forma es:

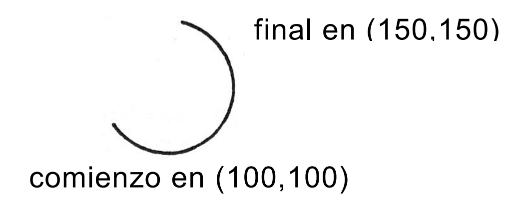
```
DRAW x,y,a
```

x e **y** se utilizan para especificar el punto final de la línea lo mismo que antes y **a** es el número de radianes que debe girarse a medida que se avanza: si **a** es un número positivo se gira a la izquierda, mientras que si **a** es negativo, se gira a la derecha. Otra forma de considerar **a** es como indicación de la fracción de un círculo completo que se dibujará: un círculo completo es 2π radianes, por lo que si **a =** π dibujará un semicírculo, si **a = 0,5***π dibujará un cuarto de un círculo y así sucesivamente.

Por ejemplo, supongamos **a = π**. Entonces, cualesquiera que sean los valores que tomen **x** e **y**, se trazarán un semicírculo. Ejecute:

10 PLOT 100,100: DRAW 50,50,PI

que dibujará lo siguiente:



El dibujo comienza en una dirección de sudeste, pero en el momento en que se detiene tiene dirección noroeste; en el recorrido entre dichos dos puntos ha girado 180° ó π radianes (el valor de **a**).

Ejecute el programa varias veces, con **PI** sustituido por otras varias expresiones, v.g. **- PI, PI/2, 3*PI/2, PI/4, 1, 0**.

La última sentencia de este capítulo es **CIRCLE**, que dibuja un círculo completo. Especifica las coordenadas del centro y del radio del círculo con el empleo de

CIRCLE coordenada x, coordenada y, radio

Lo mismo que con **PLOT** y **DRAW**, puede poner las diversas clases de elementos de color al principio de una sentencia **CIRCLE**.

La función **POINT** le dice si un pixel es de color de tinta o de papel. Tiene dos argumentos, las coordenadas del. pixel (que deben estar encerradas entre paréntesis) y su resultado es 0 si el pixel es de color de papel y 1 si es color de tinta. Pruebe:

CLS : PRINT POINT (0,0) : PLOT 0,0: PRINT POINT (0,0)

Teclee:

PAPER 7: INK 0

e investiguemos cómo **INVERSE** y **OVER** operan en el interior de una sentencia **PLOT**. Las dos afectan al pixel correspondiente y no al resto de las posiciones de carácter. Suelen estar inactivas (0) en una sentencia **PLOT**, por lo que sólo necesita mencionarlas para que se les active (1).

Se da a continuación una lista de las posibilidades para su referencia:

PLOT; -esta es la forma habitual. Traza un punto de tinta; esto es, posiciona el pixel para poner de manifiesto el color de tinta.

PLOT INVERSE 1; -traza un punto de erradicador de tinta; esto es, posiciona el pixel para poner de manifiesto el color del papel.

PLOT OVER 1: -cambia el pixel a partir de cualquier estado anterior, por lo que si tenía color de tinta adquirirá color de papel y viceversa.

PLOT INVERSE 1; OVER 1; -deja el pixel exactamente igual a como estaba antes; pero observe que también cambia la posición **PLOT**, por lo que podría utilizarle simplemente para hacer esa operación.

Como otro ejemplo de utilización de la sentencia **OVER**, llene la pantalla con escritura utilizando blanco y negro y luego, teclee:

PLOT 0,0: DRAW OVER 1;255,175

Dibujará una línea bastante correcta, aun cuando tenga interrupciones en la misma siempre que se encuentre con alguna escritura. Ahora vuelva a hacer exactamente el mismo comando. La linea se desvanecerá sin dejar ningún rastro. Esta es la gran ventaja de **OVER 1**. Si hubiera dibujado la línea utilizando

PLOT 0,0: DRAW 255,175

y la hubiera borrado con el empleo de

PLOT 0,0: DRAW INVERSE 1,255,175

entonces, habría borrado también parte de la escritura.

Ahora, pruebe:

PLOT 0,0: DRAW OVER 1;250,175

e intente borrar la línea mediante

DRAW OVER 1;-250,-175

Con esta orden no se opera de forma correcta, porque los pixels que la línea utiliza en el recorrido de retorno no son completamente los mismos que los que se emplearon en el trazado. Ha de borrar una línea en la misma dirección exactamente en que la dibujó.

Una forma de conseguir colores no habituales es introducir dos colores normales juntos en un solo cuadrado, con el empleo de un gráfico definido por el usuario. Ejecute el siguiente programa:

```
1000 FOR n = 0 TO 6 STEP 2
1010 POKE USR "a" + n, BIN 01010101: POKE USR "a" + n + 1,
        BIN 10101010
1020 NEXT n
```

que da el gráfico definido por el usuario que corresponde a una configuración de

tablero de ajedrez. Si imprime este carácter (modo de gráfico, luego **a**) en tinta roja sobre papel amarillo, le encontrará con un color naranja moderadamente aceptable.

Ejercicios

1. Juegue con los elementos **PAPER**, **INK**, **FLASH** y **BRIGHT** en una sentencia **PLOT**. Estas son las partes que afectan a la totalidad de la posición de carácter que contiene el pixel. Normalmente es como si la sentencia **PLOT** hubiera comenzado

   ```
   PLOT PAPER 8; FLASH 8; BRIGHT 8;...
   ```

   y sólo se altera el color de tinta de una posición de carácter cuando algo se traza en la misma, pero puede cambiarlo si lo desea. Tenga especial cuidado cuando utilice colores con INVERSE 1, porque posiciona el pixel para poner de manifiesto el color del papel, pero cambia el color de la tinta y éste podría no ser lo que esperaba.

2. Intente dibujar círculos con el empleo de **SIN** y **COS** (si ha leído el capítulo 9, trate de resolverlo). Ejecute el programa siguiente:

   ```
   10 FOR n=0 TO 2*PI STEP PI/180
   20 PLOT 100+80*COS n,87+80*SIN n
   30 NEXT n
   40 CIRCLE 150,87,80
   ```

   Puede ver que la sentencia CIRCLE es mucho más rápida, aunque menos exacta.

3. Pruebe:

   ```
   CIRCLE 100,87,80: DRAW 50,50
   ```

A partir de ella puede constatar que la sentencia **CIRCLE** deja la posición **PLOT** en un lugar bastante indeterminado (siempre está en algún lugar a medio camino ascendente en el lado derecho del círculo. Habitualmente necesitará seguir a la sentencia **CIRCLE** con una sentencia **PLOT** antes de que haga cualquier otro dibujo.

4. Veamos un programa para dibujar la gráfica de casi cualquier función. Primero, le pide un número **n**; trazará la gráfica para valores desde **-n** a **+n**. A continuación, le pide la función propiamente dicha, introduciéndola como una cadena, La cadena debe ser una expresión que utilice **x** como el argumento de la función.

```
 10 PLOT 0,87: DRAW 255,0
 20 PLOT 127,0: DRAW 0,175
 30 INPUT s,e$
 35 LET t=0
 40 FOR F =0 TO 255
 50 LET x = (f-128)*s/128: LET y = VAL e$
 60 IF ABS y >87 THEN LET t=0: GO TO 100
 70 IF NOT t THEN PLOT f,y+88: LET t=1: GO TO 100
 80 DRAW 1,y- antiguay
100 LET antiguay = INT (y +x)
110 NEXT F
```

Ejecute el anterior programa, a título de ejemplo, introduzca por el teclado **10** para el número **n** y **10*TAN x** para la función. Trazará una gráfica de tangente de **x**, cuando **x** varía desde -10 a +10.

CAPITULO 18

Movimiento

Resumen
PAUSE, INKEYS, PEEK

Con bastante frecuencia, deseará conseguir que el programa tenga una duración determinada y para tal fin, encontrará de utilidad la sentencia **PAUSE**.

PAUSE n

interrumpe el cómputo y congela el contenido de pantalla durante *n* imágenes de la televisión (a 50 imágenes por segundo en Europa o 60 en América). El valor *n* puede ser de hasta 65535, lo que le proporciona poco menos de 22 minutos; si n=0, entonces, significa «**Pausa** para siempre».

Una pausa siempre puede acortarse pulsando una tecla (tenga presente que un espacio con **CAPS SHIFT** producirá también una interrupción). Ha de pulsar la tecla después de que haya comenzado la pausa.

El siguiente programa actúa sobre el segundero de un reloj.

```
 10 REM Primero dibujamos el cuadrante del reloj
 20 FOR n = 1 TO 12
 30 PRINT AT 10-10*COS (n/6* PI),16+10*SIN(n/6*PI);n
 40 NEXT n
 50 REM Ahora ponemos en marcha el reloj
 60 FOR t =0 TO 200000: REM t es el tiempo en segundos
 70 LET a=t/30*PI : REM a es el ángulo del segundero en radianes
 80 LETsx=80*SIN a: LET sy = 80*COS a
200 PLOT 128,88: DRAW OVER 1;sx,sy: REM dibujo del segundero
210 PAUSE 42
220 PLOT 128,88: DRAW OVER 1; sx,sy: REM borrar segundero
400 NEXT t
```

La «cuerda» de este reloj se acabará en unas 55,5 horas debido a la línea 60, pero fácilmente puede prolongar su período de funcionamiento. Observe cómo la temporización está controlada por la línea 210. Podría esperar **PAUSE 50** para hacerle dar un «tic-tac» de un segundo, pero el cómputo lleva algo de tiempo también y ha de permitírselo. Esto es mejor hacerlo por tanteo, sincronizando el reloj del ordenador con uno real y ajustando la línea 210 hasta que concuerden (no puede hacer esta operación con una gran exactitud; un ajuste de un cuadro en un segundo es el 2% o la mitad de una hora en un día).

Hay una manera mucho más exacta de medir el tiempo, utilizando el contenido de determinadas posiciones de memoria. Los datos almacenados se recuperan con el empleo de **PEEK**. En el capítulo 25 se explicará con detalle. La expresión utilizada es:

(65536*PEEK 23674+ 256*PEEK 23673+ PEEK 23672)/50

Nos da el número de segundos desde que el ordenador se puso en marcha (hasta unos 3 días y 21 horas, cuando retorna a 0).

Damos a continuación, un programa de reloj para aplicar lo anterior:

```
 10 REM Primero, dibujarnos el cuadrante del reloj
 20 FOR n = 1 TO 12
 30 PRINT AT 10-10* COS (n/6*PI),16+10*SIN(n/6*PI);n
 40 NEXT n
 50 DEF FN t() = INT ((65536*PEEK 23674+256*PEEK 23673+
    PEEK 23672)/50): REM número de segundos desde puesta en
    marcha
100 REM Ahora ponemos en marcha el reloj
110 LET t1= FN t()
120 LET a = t1/30*PI : REM a es el ángulo del segundero en radianes
130 LET sx = 72*SIN a: LET sy = 72*COS a
140 PLOT 131,91: DRAW OVER 1;sx,sy: REM dibujo segundero
200 LET t = FN t()
210 IF t <= t1 THEN GO TO 200: REM esperar hasta tiempo para
    siguiente segundero
220 PLOT 131,91: DRAW OVER 1;sx,sy: REM borrar anterior
    segundero
230 LET t1=t: GO TO 120
```

El reloj interno que utiliza este método debe tener una exactitud de un 0,01 % mientras el ordenador esté ejecutando su programa, o lo que es lo mismo, 10 segundos por día; pero se detiene temporalmente siempre que haga **BEEP**, o una operación de cinta de cassette, o utilice !a impresora o cualquiera de los demás elementos suplementarios de equipos que pueda emplear con el ordenador. Todas estas operaciones le harán perder tiempo.

Los números **PEEK 23674**, **PEEK 23673** y **PEEK 23672** se retienen en el interior del ordenador y se utilizan para el contaje en 1/50 de segundo. Cada uno está entre 0 y 255 y se incrementan gradualmente a través de todos los números desde 0 a 255; después de 255 pasan directamente a 0.

El que se incrementa más frecuentemente es **PEEK 23672**. Cada 1/50 de segundo se incrementa en 1. Cuando está en 255, el siguiente incremento le lleva a 0 y, al mismo tiempo, impulsa a **PEEK 23673** hasta 1. Cuando (cada 250/50 segundos) **PEEK 23673** se lleva desde 255 a 0, impulsa, a su vez, a **PEEK 23674** en 1. Esto debe ser suficiente para explicar por qué opera adecuadamente la anterior expresión.

Ahora, veámoslo detenidamente. Suponga que nuestros tres números son 0 (para **PEEK 23674**), 255 (para **PEEK 23673**) y 255 (para **PEEK 23672**). Ello significa que al cabo de unos 21 minutos después del encendido, nuestra expresión debiera proporcionar:

(65536 *0 + 256 *255 + 255)/50 =1310,7.

Pero hay un peligro oculto. La siguiente vez que haya un contaje de 1/50 segundo, los tres números cambiarán a 1, 0 y 0. Frecuentemente, esto sucederá cuando esté a medio camino de la evaluación de la expresión: el ordenador evaluaría **PEEK 23674** como 0, pero, entonces, cambian los otros dos a 0 antes de que pueda aplicarles la función **PEEK**. La respuesta sería, entonces:

(65536*0 + 256 *0 + 0) / 50 = 0

que está desesperadamente equivocada.

Una sencilla regla para evitar este problema es *evaluar la expresión dos veces en sucesión y tomar la respuesta más grande*.

Si examina detenidamente el anterior programa puede ver que cumple con dicha regla de forma implícita.

Veamos un artificio para aplicar la regla. Defina las funciones:

```
10 DEF FN m(x,y)=(x+y+ABS(x-y))/2: REM la más grande de
    x y de y
20 DEF FN u()=(65536*PEEK 23674+256*PEEK 23673+PEEK
   23672)/50: REM tiempo, puede estar equivocado
30 DEF FN t()= (FN u()): REM tiempo, correcto
```

Puede cambiar los tres números del contador de modo que proporcionen el tiempo real en lugar del tiempo desde que el computador se puso en marcha. Por ejemplo, para ajustar el tiempo a las 10,00 de la mañana, se percatará de que equivale a 10*60*60*50 = 1800000 cincuentavas partes de un segundo y que:

1800000 = 65536*27 + 256*119 + 64

Para poner los tres números a 27, 119 y 64, hace:

```
POKE 23674,27: POKE 23673,119: POKE 23672,64
```

En los países con frecuencias de red de 60 hertzios, estos programas deben sustituir 50 por 60, en donde sea adecuado.

La función **INKEY**$ (que no tiene ningún argumento) efectúa la lectura del teclado. Si está pulsando exactamente una tecla (o una tecla **SHIFT** y cualquier otra tecla), entonces, el resultado es el carácter que dicha tecla da en modo L; de no ser así, en cualquier otro modo, el resultado es la cadena vacía.

Pruebe el siguiente programa que opera como una máquina de escribir:

```
10 IF INKEY$ <> "" THEN GO TO 10
20 IF INKEY$= "" THEN GO TO 20
30 PRINT INKEY$;
40 GO TO 10
```

En este caso, la línea 10 espera a que retire su dedo del teclado y la línea 20 espera a que pulse una nueva tecla.

Recuerde que a diferencia con **INPUT**, **INKEY**$ no le espera. Por ello no teclee **ENTER**, pero, por otra parte, si no teclea nada en absoluto, entonces, habrá perdido su oportunidad.

Ejercicios

1. ¿Qué sucede si omite la línea 10 en el programa de máquina de escribir?

2. Otra forma de utilizar INKEY$ es una conjunción con PAUSE, como en este programa de máquina de escribir de alternativa.

```
10 PAUSE 0
20 PRINT INKEY$;
30 GO TO 10
```

Para hacer este trabajo ¿por qué es esencial que una pausa no deba acabar si le encuentra pulsando una tecla cuando da comienzo?

3. Adaptar el programa del segundero de modo que indique también el minutero y la aguja de las horas, dibujándolos cada minuto. Si se considera ambicioso, disponga que cada cuarto de hora haga cualquier «efecto especial' (por ejemplo, podría hacer que diera las campanadas del Big Ben con **BEEP**). Vea el siguiente capítulo.

4. (Para sádicos). Pruebe este programa:

```
10 IF INKEY$="" THEN GO TO 10
20 PRINT AT 11,14;"OUCH!"
30 IF INKEY$ <> "" THEN GO TO 30
40 PRINT AT 11,14"     "
50 GO TO 10
```

CAPITULO 19

BEEP

Resumen
BEEP

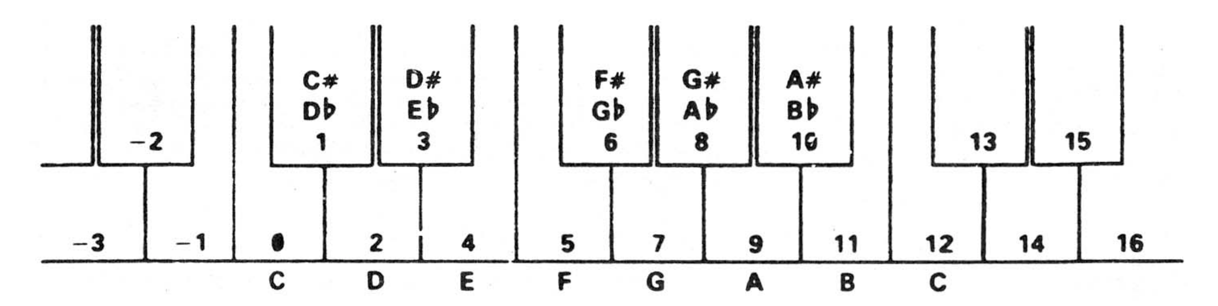
Si no ha descubierto todavía que el ZX Spectrum tiene un altavoz incorporado, lea el fascículo de introducción antes de proseguir.

El altavoz se hace sonar con el empleo de la sentencia **BEEP**

BEEP duración, tono

en donde, como es habitual, «duración» y «tono» representan cualquier expresión numérica. La duración se da en segundos y el tono se da en semitonos por encima del «do» central (C) - empleando números negativos para las notas por debajo del «do» central.

Damos un diagrama para mostrar los valores del tono de todas las notas en una octava en el piano:



C=do, D=re, E=mi, F=fa, G=sol, A=la, B=si
#=sostenido b=bemol

Para conseguir notas más altas o más bajas, ha de sumar o restar 12 para cada octava que suba o baje.

Si tiene un piano frente a usted cuando esté programando una armonía, este diagrama probablemente será todo lo que necesite para obtener los valores del tono.

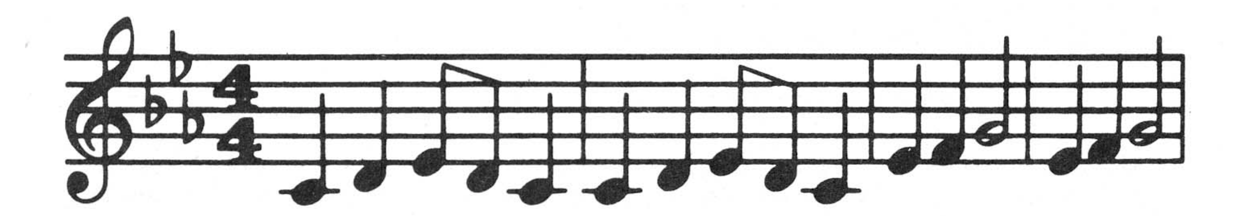
Si, sin embargo, está transcribiendo directamente de alguna música escrita, entonces, le sugerimos que dibuje un diagrama del pentagrama con el valor del tono escrito contra cada línea y espacio, teniendo en cuenta la tecla.

Por ejemplo, teclee:

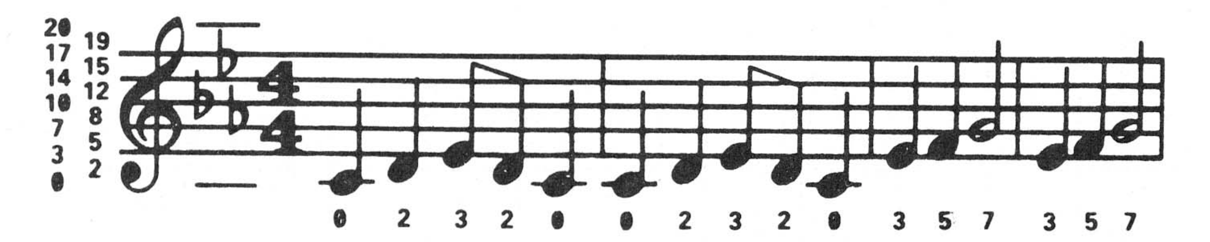
```
10 PRINT "Frere Jacques"
20 BEEP 1,0: BEEP 1,2: BEEP.5,3: BEEP. 5,2: BEEP 1,0
30 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP 1,0
40 BEEP 1,3: BEEP 1,5: BEEP 2,7
50 BEEP 1,3: BEEP 1,5: BEEP 2,7
60 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3:
   BEEP .5,2: BEEP 1,0
70 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3:
   BEEP.5,2: BEEP 1,0
80 BEEP 1,0: BEEP 1,-5: BEEP 2,0
90 BEEP 1,0: BEEP 1,-5: BEEP 2,0
```

Cuando lo ejecute, debe obtener la marcha fúnebre de la primera sinfonía de Mahler, el fragmento en que los duendes entierran el hombre de la caballería de los Estados Unidos.

Suponga, por ejemplo, que su composición está escrita en la clave de «do» en tono menor, como la sinfonía de Mahler anterior. La entrada es como se indica a continuación.



y puede escribir los valores del tono de las notas como:



Hemos puesto dos líneas suplementarias, sólo las necesarias. Observe cómo la nota «mi» baja en la armadura afecta no solamente a la nota «mi» en el espacio superior, bemolizándole de 16 a 15, sino también la nota «mi» en la línea inferior, bemolizándola de 4 a 3. Ahora debe ser bastante fácil encontrar el valor del tono de cualquier nota en el pentagrama.

Si desea cambiarla clave de la pieza, lo mejor que puede hacerse es establecer una clave, **clave**, variable e insertar **clave +** antes de cada valor de tono; así, la segunda línea se hace:

```
20 BEEP 1,key+0: BEEP 1,key+2: BEEP .5,key+3: BEEP.5,
   key + 2: BEEP 1,key+0
```

Antes de que ejecute un programa debe dar a **key** el valor adecuado: O para «do» en tono menor, 2 para «re» en tono menor, 12 para «do» en tono menor una octava superior y así sucesivamente. Puede conseguir afinar el ordenador con otro instrumento ajustando **key**, empleando valores fraccionarios.

También puede establecer las duraciones de todas las notas. Puesto que se trata de una pieza bastante lenta, hemos fijado un segundo para una negra y el resto se basan en ello, medio segundo para una corchea y así sucesivamente.

Es más flexible establecer una variable, **crochet**, para almacenar la longitud de una negra y especificar las duraciones en función de ella. A continuación, la línea 20 se haría:

```
20 BEEP crotchet,key + 0: BEEP crotchet, key+2: BEEP crotchet/
   2,key + 3: BEEP crotchet/2,key + 2: BEEP crotchet,key + 0
```

(probablemente deseará dar a **crotchet** y a **key** nombres más cortos).

Dando a la negra **crotchet**, valores adecuados, fácilmente puede variar la velocidad de la pieza.

Recuerde que puesto que hay solamente un altavoz en el ordenador, sólo puede tocar una nota a la vez, por lo que está restringido a composiciones no polifónicas. Si desea algo más debe cantar por su cuenta.

Intente programar melodías por si mismo, comenzando con una bastante sencilla como la de «Tres ratones ciegos». Si no tiene piano ni nunca ha escrito música, apodérese de un instrumento muy sencillo como un silbato o un caramillo y trate de componer melodías. Podría hacer un diagrama que muestre el valor del tono para cada nota que pueda tocar en este instrumento:

Teclee:

```
FOR n=0 to 1000: BEEP .5,n: NEXT n
```

Tocará notas tan altas como pueda y luego se interrumpirá con el mensaje de error **B integer out of range**. Puede imprimir n para averiguar cuán altas fueron las notas conseguidas.

Pruebe lo mismo pero bajando las notas. Las notas muy bajas sonarán como chasquidos; de hecho, las notas más altas también están constituidas por chasquidos de la misma forma, pero más rápidos, por lo que el oído no puede distinguirlos.

Solamente la gama media de notas son realmente buenas para la música; las notas bajas suenan demasiado como chasquidos y las notas altas son agudas y tienden a gorjear un poco.

Introduzca por el teclado la línea de programa siguiente:

```
10 BEEP .5,0: BEEP .5,2: BEEP .5,4: BEEP .5,5: BEEP .5,7:
   BEEP .7,9: BEEP .5,11: BEEP .5,12: STOP
```

Con ella se toca la escala de «do» en tono mayor, que utiliza todas las notas «blancas» en el piano desde la nota «do» media a la siguiente nota «do» arriba. La forma en que esta escala se armoniza es exactamente la misma que en un piano y se denomina afinación plácida, porque el intervalo de altura de un semitono es el mismo a lo largo de toda la escala. Sin embargo, un violinista tocaría la escala de forma muy poco diferente, ajustando todas las notas para hacerlas sonar más agradable al oído. Puede hacerlo moviendo sus dedos muy ligeramente hacia arriba o hacia abajo en las cuerdas de una forma que un pianista no puede hacer.

La escala natural, que es lo que toca el violinista, se obtiene con:

```
20 BEEP .5,0: BEEP .5,2.039: BEEP .5,3.86: BEEP .5,4.98:
   BEEP .5,7.02: BEEP .5,8.84: BEEP .5,10.88: BEEP .5,12: STOP
```

Quizá pueda, o no pueda, ser capaz de detectar cualquier diferencia entre estas dos; algunas personas sí pueden. La primera diferencia notable es que la tercera nota es algo más baja en la escala naturalmente afinada. Si usted es un perfeccionista real, podría desear programar sus armonías para utilizar esta escala natural en lugar de la plácida antes adoptada. La desventaja es que aunque actúa perfectamente

en la clave de la nota «do», en otras no sucede lo mismo (todas ellas tienen sus propias escalas naturales) y en algunas, lo hace muy mal. La escala uniformemente temperada (plácida) está solamente algo desactivada y opera igualmente bien en todas las claves o tonos.

Esto es menos problema en el ordenador, por supuesto, porque puede utilizar el artilugio de añadir una clave, **key**, variable.

Alguna música, especialmente la música india, utiliza intervalos de altura inferior a un semitono. Puede programarlos en una sentencia **BEEP** sin ninguna dificultad; por ejemplo, el cuarto de tono por encima de la nota «do» media tiene un valor de altura acústica de 0,5.

Puede hacer que el teclado suene como una bocina, en vez de con chasquidos, mediante:

POKE 23609,255

El segundo número determina la longitud del sonido de bocina (pruebe con diversos valores entre 0 y 255). Cuando es 0, el sonido es tan corto que parece un chasquido suave.

Si está interesado por hacer más con el sonido del Spectrum, como oír el sonido que **BEEP** produce en algo distinto al altavoz interno, encontrará que la señal está presente en las conexiones del micrófono y del auricular. estará a un nivel más alto en la de los auriculares, pero, de cualquier modo, son las mismas. Puede usar esta característica para conectar un auricular o un par de auriculares a su Spectrum. Con ello no se cortará el altavoz interno. Si está realmente ansioso de producir mucho ruido podría conectarle hasta un amplificador (la conexión «MIC» probablemente le proporcionará el nivel correcto) o podría registrar el sonido en una cinta y conseguir que el Spectrum lo toque conjuntamente.

No deteriorá al Spectrum aunque cortocircuite las conexiones «MIC» o «EAR», por lo que debe experimentar para encontrar la mejor prestación para lo que quiera hacer.

Ejercicio

1. Volver a escribir el programa de la sinfonía de Mahler del capítulo 8, de modo, que utilice los bucles **FOR** para repetir las rayas del compás.

   Programe el ordenador de modo que toque no solamente la marcha fúnebre, sino también el resto de la primera sinfonía de Mahler.

CAPITULO 20

Almacenamiento en cinta

Resumen
LOAD, SAVE, VERIFY, MERGE

Los métodos básicos para utilizar la grabadora de cassette para conservar (**SAVE**), cargar (**LOAD**) y verificar (**VERIFY**) programas se dan en el fascículo de introducción. Esta sección debe ser objeto de lectura e intentarse realizar los procedimientos antes de seguir más adelante.

Hemos visto que LOAD borra el anterior programa y las variables existentes en el ordenador antes de cargar los nuevos procedentes de la cinta; hay otra sentencia, MERGE, que no hace lo mismo. MERGE sólo borra una variable o línea de programa anterior si ha de hacerlo porque haya una nueva con el mismo nombre o número de línea. Introduzca por el teclado el programa de «dados» incluido en el capítulo 11 y consérvelo en cinta, como «dice» (dados). Ahora introduzca y ejecute lo siguiente:

```
 1 PRINT 1
 2 PRINT 2
10 PRINT 10
20 LET x = 20
```

y luego, proceda como para la verificación, pero sustituyendo **VERIFY** «dice» por:

```
MERGE "dice"
```

Si efectúa el listado del programa, puede ver que han sobrevivido las líneas 1 y 2, pero que las líneas 10 y 20 se han sustituido por las del programa de dados. **x** también ha sobrevivido (pruebe **PRINT x**).

Ya ha visto formas sencillas de las cuatro sentencias utilizadas con la cinta de cassette:

SAVE registra el programa y las variables en cassette
VERIFY comprueba el programa y las variables en cassette con respecto a los ya existentes en el ordenador
LOAD borra del ordenador todos sus programas y variables anteriores y les sustituye por otros nuevos extraídos de cassette.
MERGE es como **LOAD** con la salvedad de que no borra una variable o línea de programa anterior, a no ser que haya de hacerlo porque su nombre o número de línea sea coincidente con una nueva procedente del cassette.

En cada una de ellas, la palabra clave va seguida por una cadena; para **SAVE** ello proporciona un nombre para el programa en cinta, mientras que para las otras tres, comunica al ordenador qué programa buscar. Mientras realiza la búsqueda, imprime el nombre de cada programa que encuentra. Hay un par de peculiaridades interesantes para todo ello.

Para **VERIFY**, **LOAD** y **MERGE**, puede proporcionar la cadena vacía como el nombre para buscar; entonces, el ordenador no tiene que preocuparse del nombre, sino que toma el primer programa que se encuentra.

Una variante en **SAVE** adopta la forma:

SAVE cadena **LINE** número

Un programa conservado con su empleo se registra de tal forma que cuando sea objeto de nueva lectura por **LOAD** (pero no **MERGE**) salta automáticamente a la línea con el número dado, con la que se ejecuta por sí misma.

Hasta ahora, los únicos tipos de información que hemos almacenado en cassette ha sido programas junto con sus variables. Hay, asimismo, otras dos tipos denominados *matrices* y *bytes*.

Las matrices se tratan con pocas diferencias.

Puede conservar matrices en cinta con el empleo de DATA en una sentencia **SAVE**, mediante:

SAVE cadena DATA nombre de matriz()

La **cadena** es el nombre que la información tendrá en la cinta y opera en exactamente la misma forma que cuando conserva un programa o bytes ordinarios.

El nombre de la matriz especifica la matriz que desea conservar, por lo que es simplemente una letra seguida por $. Recuerde los paréntesis en lo sucesivo; podría considerar que son lógicamente innecesarios pero, no obstante, ha de ponerlos para hacerlo más fácil para el ordenador.

Están claras las funciones independientes de la cadena y del *nombre de la matriz*. Si dice, por ejemplo:

SAVE "Bloggs" DATA b()

entonces, **SAVE** toma la matriz *b* del ordenador y la almacena en cinta con el nombre «Bloggs". Cuando teclea:

VERIFY "Bloggs" DATA b()

el ordenador buscará una matriz de números almacenada en cinta con el nombre «Bloggs» (cuando la encuentre, escribirá «Matriz de números: Bloggs») y la comprobará con respecto a la matriz b en el ordenador.

LOAD "Bloggs" DATA b()

encuentra la matriz en la cinta y luego (si hay espacio para ella en el ordenador) borra cualquier matriz ya existente, denominada *b* y carga la nueva matriz procedente de la cinta, a la que llama *b*.

No puede utilizar **MERGE** con matrices almacenadas.

Puede almacenar matrices de caracteres (cadenas) de la misma forma. Cuando el ordenador está buscando la cinta y encuentra una de estas, escribe «Matriz de caracteres»: seguida por el nombre. Cuando carga una matriz de caracteres, no borrará solamente cualquier matriz de caracteres anterior con el mismo nombre, sino también cualquier cadena con el mismo nombre.

El almacenamiento de bytes se utiliza para fragmentos de información sin ninguna referencia a lo que se aplica la información que podría ser: una

imagen de televisión o gráfico definido por el usuario o algo que haya elaborado por si mismo. Se muestra con el empleo de la palabra **CODE**, como en:

SAVE 'imagen" CODE 16384,6912

La unidad de almacenamiento en memoria es el byte (un número entre 0 y 255) y cada byte tiene una dirección (que es un número entre 0 y 65535). El primer número después de CODE es la dirección del primer byte que ha de almacenarse en cinta y el segundo es el número de bytes a almacenar. En nuestro caso, 16384 es la dirección del primer byte en el fichero de presentación visual (que contiene la imagen de televisión) y 6912 es el número de bytes en la misma, por lo que estamos conservando una copia de la pantalla de televisión (haga la prueba). El nombre **imagen»** opera lo mismo que los nombres para programas.

Para volverle a cargar, utilice:

LOAD "imagen" CODE

Puede poner números después de CODE en la forma:

LOAD nombre **CODE** comienzo, longitud

En este caso, la *longitud* es simplemente una medida de seguridad; cuando el ordenador ha encontrado los bytes en cinta con el nombre correcto, no obstante, rehusará cargarles si hay más que la *longitud* de los mismos, puesto que, evidentemente, hay más datos de los que esperaba y podría el ordenador sobreescribir algo que no tuviera usted intención de modificar. Dará el informe de error **R Tape loading error**. Puede omitir *longitud* y, entonces, el ordenador efectuará la lectura de cuantos bytes haya.

Comienzo indica la dirección en donde ha de volverse a cargar el primer byte y que puede ser diferente de la dirección en la que fue conservado, aunque si son las mismas puede omitir *comienzo* en la sentencia **LOAD**.

CODE 16384,6912 es tan útil para conservar y cargar la imagen que puede sustituirle por **SCREEN$**, por ejemplo:

SAVE "imagen" SCREEN$
LOAD "imagen" SCREEN$

Este es un caso raro para el que no operará **VERIFY**. Tenga en cuenta que **VERIFY** escribe los nombres de lo que encuentra en la cinta, de modo que en el momento en que llega a la verificación se ha cambiado el fichero de la presentación visual y falla la verificación. En todos los demás casos, debe utilizar **VERIFY** siempre que emplee **SAVE**.

A continuación se da un resumen de las cuatro sentencias utilizadas en este capitulo.

Nombre significa cualquier expresión de cadena y se refiere al nombre con el que se almacena la información en cassette. Debe estar constituido por los caracteres de impresión en ASCII, de los que sólo se utilizan los 10 primeros.

Hay cuatro clases de información que se pueden almacenar en cinta: programa y variables (juntas), matrices de números, matrices de caracteres y bytes puros.

Cuando **VERIFY**, **LOAD** y **MERGE** están buscando información en la cinta con un nombre dado y de una clase dada, imprimen en la pantalla la clase y el nombre de toda la información que encuentran. La clase se indica por «Programa:», «Matriz de números:», «Matriz de caracteres:» o «Bytes:». Si nombre era la cadena vacía, toman el primer lote de información de la clase adecuada, haciendo caso omiso de su nombre.

SAVE

Conserva información en cinta bajo el nombre dado. El error F se produce cuando el nombre está vacío o tiene 11 o más caracteres.

SAVE siempre pone un mensaje **«Start tape, then press any key»** (ponga en marcha cinta, luego pulse cualquier tecla) y espere a que se pulse una tecla antes de conservar algo.

1. Programa y variables:

SAVE nombre

es la forma normal.

SAVE nombre **LINE** número de línea

conserva el programa y variables de tal forma que la carga **LOAD** se sigue automáticamente con

GO TO número de línea

SAVE nombre **LINE** es equivalente a **SAVE** nombre **LINE 1**, de modo que cuando se cargue el programa, se ejecute por sí mismo desde el principio.

2. Bytes:

SAVE nombre **CODE** comienzo, longitud

conserva la longitud en bytes que comienza en la dirección *comienzo*.

SAVE nombre **SCREEN$**

es equivalente a

SAVE nombre **CODE** 16384,6912

y conserva la imagen de televisión.

3. Matrices:

SAVE nombre **DATA** letra()

o

SAVE nombre **DATA** letra $()

conserva la matriz cuyo nombre es *letra* o *letra$* (ello exige no tener ninguna relación con nombre).

VERIFY

Comprueba la información en cinta con respecto a la información existente en memoria. La falta de verificación da el informe de error **R Tape loading error** (error de carga de cinta).

1. Programa y variables:

VERIFY nombre

2. Bytes:

VERIFY nombre **CODE** comienzo , longitud

Si los bytes de nombre en cinta son más en número que en *longitud*, entonces, da el error R. De no ser así, comprobarles con respecto a los bytes en memoria que comienzan en la dirección comienzo.

VERIFY nombre **CODE** comienzo

comprueba los bytes de nombre en cinta con respecto a los que hay en memoria que comienzan en la dirección comienzo.

VERIFY nombre **CODE**

comprueba los bytes en nombre en la cinta con respecto a los que hay en memoria que comienza en la dirección a partir de la cual se conservó el primer byte de cassette.

VERIFY nombre **SCREEN$**

es equivalente a:

VERIFY nombre **CODE 16384,6912**

pero, casi con toda certeza, dejará de verificar.

3. Matrices:

VERIFY nombre **DATA** letra ()

o

VERIFY nombre **DATA** letra $ ()

comprueba el **nombre** de matriz con respecto a la *letra* o *letra$* de matriz en memoria.

LOAD
carga nueva información a partir de la cinta, borrando de la memoria la anterior información.

1. Programa y variables:

LOAD nombre

borra el anterior programa y variables y carga el nombre de variables y programa a partir de la cassette; si el programa se conservó utilizando **SAVE** nombre **LINE**, el ordenador realiza un salto automático.

El error **4 Out of memory** se produce si no hay espacio para el nuevo programa y variables. En este caso, no se borran el anterior programa y variables.

2. Bytes:

LOAD nombre **CODE** comienzo, longitud

Si los bytes en *nombre* de la cinta son más en número que en longitud, entonces da un error R. De no ser así, los carga en memoria empezando en la dirección *comienzo* y recubriendo cualquier cosa que hubiera anteriormente.

LOAD nombre **CODE** comienzo

carga *nombre* en bytes desde cinta en memoria, empezando en la dirección *comienzo* y recubriendo cualquier cosa que hubiera con anterioridad.

LOAD nombre **CODE**

carga el *nombre* en bytes desde cinta en meoria empezando en la dirección a partir de la cual se conservó el primer byte de cinta y recubriendo los bytes que estaban allí en memoria con anterioridad.

3. Matrices:

LOAD nombre **DATA** letra ()

o

LOAD nombre **DATA** letra $ ()

borra cualquier matriz ya denominada *letra* o *letra $* (según sea adecuado) y forma una nueva a partir de la matriz almacenada en cassette.

El error **4 Out of memory** se produce si no hay espacio para matrices nuevas. No se borran las matrices anteriores.

MERGE

Carga la nueva información desde cassette sin borrar la anterior información de la memoria.

1. Programa y variables:

MERGE nombre

funde el **nombre** del programa con el ya existente en memoria, recubriendo cualesquiera variables o líneas de programa en el anterior programa cuyos nombres o números de línea no están en concordancia con los del nuevo programa.

El error **4 Out of memory** se produce a no ser que haya suficiente espacio en memoria para la totalidad del anterior programa y variables y la totalidad del nuevo programa y variables que se cargan desde la cinta.

2. Bytes:

Imposible

3. Matrices:

Imposible

Ejercicios

1. Haga una grabación en cassette en la que el primer programa, cuando se carga, imprime un *menú* (una lista de los demás programas en la cassette), le pida que elija un programa y luego, lo carga.

2. Tome el gráfico de piezas de ajedrez del capítulo 14 y luego, teclee **NEW** y se seguirán conservando. Sin embargo, no se conservarán cuando se desconecte el ordenador; si desea mantenerles, debe conservarlos en cita, utilizando **SAVE** con **CODE**. La forma más fácil de conservar los veintiún gráficos definidos por el usuario mediante:

SAVE "ajedrez" CODE USR "a",21 *8

seguida por:

VERIFY "ajedrez" CODE

Este es el sistema de conservación de *bytes* que se empleó para la conservación de la imagen. La dirección del primer byte a conservar es **USR "a"**, la dirección

del primero de los ocho bytes que determinan la configuración del primer gráfico definido por el usuario y el número de bytes a conservar es 21*8 (8 bytes por cada uno de los 21 gráficos).

Para una nueva carga, normalmente utilizaría

LOAD "ajedrez" CODE

Sin embargo, si están volviendo a cargar en un Spectrum con una cantidad de memoria diferente o si han desplazado el gráfico definido por el usuario a una dirección distinta (ha de hacer esto deliberadamente con el empleo de técnicas más avanzadas), ha de tener más cuidado y utilizar:

LOAD "ajedrez" CODE USR "a"

USR permite el hecho de que el gráfico debe cargarse de nuevo en una dirección diferente.

CAPITULO 21

La Impresora ZX

Resumen
LPRINT, LLIST, COPY

Nota. Ninguna de estas sentencias es de BASIC estándar, aunque **LPRINT** se utiliza por otros ordenadores.

Si dispone de una impresora ZX, tendrá algunas instrucciones de su manejo. En este capítulo se estudian las sentencias BASIC necesarias para hacerla trabajar.

Las dos primeras, **LPRINT** y **LLIST**, son lo mismo que **PRINT** y **LIST**, con la salvedad de que utilizan la impresora en vez de la televisión. (La L inicial es un «accidente histórico» pues cuando se inventó el lenguaje BASIC se acostumbraba a emplear una máquina de escribir eléctrica en lugar de una televisión, por lo que **PRINT** significaba realmente impresión. Si deseara gran cantidad de datos a la salida, tendría que utilizar una impresora de línea muy rápida unida al ordenador y una sentencia LPRINT con el significado de Impresora de línea **PRINT').**

Pruebe el siguiente programa a título de ejemplo:

```
10 LPRINT "Este programa".'
20 LLIST
30 LPRINT ' "imprime el juego de caracteres" '
40 FOR n = 32 TO 255
50 LPRINT CHR$ n;
60 NEXT n
```

La tercera sentencia, **COPY**, imprime una reproducción de la pantalla de televisión. Por ejemplo, teclee **LIST** para conseguir un listado en la pantalla del programa anterior y teclee:

```
COPY
```

Observe que **COPY** no actúa con uno de los istados que el ordenador realiza automáticamente, porque se suprime siempre que se obedezca una orden o comando. Debe utilizar primero **LIST** o bien emplear **LLIST** y prescindir de **COPY**.

Siempre puede parar la impresora, cuando esté funcionando, al pulsar la tecla **BREAK** (**CAPS SHIFT** y **SPACE**).

Si ejecuta estas sentencias sin la impresora conectada, se perderá toda la salida y se proseguirá con la siguiente sentencia.

Pruebe el programa siguiente:

```
10 FOR n = 31 TO 0 STEP -1
20 PRINT AT 31-n,n; CHR$ (CODE "0" + n);
30 NEXT n
```

Verá una configuración de caracteres que bajan diagonalmente desde la esquina superior derecha hasta que alcanza la parte inferior de la pantalla, entonces el pro-

grama pregunta si desea un desplazamiento hacia arriba en forma de rodillo («scrolling»).

Ahora cambie **AT 31-n,n** en la línea 20 por **TAB n**. El programa tendrá exactamente el mismo efecto que antes.

Ahora cambie **PRINT** en la línea 20 por **LPRINT**. Esta vez no habrá ninguna pregunta de **scroll?**, que no debe producirse con la impresora y la configuración se mantendrá directamente con las letras F a O.

Ahora cambie **TAB n** por **AT 31-n,n**, utilizando todavía **LPRINT**. Esta vez obtendrá una sola línea de símbolos. La razón para la diferencia es que la salida por **LPRINT** no se imprime de forma directa, sino que se dispone en una memoria intermedia una imagen de una sola línea que el ordenador transmitirá a la impresora cuando llegue a ella. La impresión tiene lugar:

(i) cuando la memoria intermedia está llena
(ii) después de una sentencia **LPRINT** que no finalice en una coma o punto y coma
(iii) cuando una coma, apóstrofe o elemento **TAB** requiere una nueva línea o
(iv) al final de un programa, si se ha dejado algo sin imprimir.

El apartado (iii) explica la razón por la que nuestro programa con **TAB** actúa en la forma en que lo hace. Como para **AT**, se ignora el número de línea y la posición de **LPRINT** (como la posición **PRINT**, pero para la impresora en lugar de la televisión) se cambia al número de columna. Un elemento AT nunca puede hacer que una línea se envíe a la impresora.

Ejercicio

1. Hacer una representación gráfica impresa de **SIN** ejecutando el programa del capítulo 19 y utilizando, luego, **COPY**.

CAPITULO 22

Otros equipos accesorios

Hay otros accesorios que podrá conectar al Spectrum.

El ZX Microdrive es un dispositivo de almacenamiento masivo de alta velocidad y es mucho más flexible en la forma en que puede utilizarse que una grabadora cassette. Funcionará no solamente con **SAVE**, **VERIFY**, **LOAD** y **MERGE**, sino también con **PRINT**, **LIST**, **INPUT** y **INKEY$**.

La red se utiliza para conectar varios Spectrum, de modo que puedan comunicarse entre sí (uno de los empleos posibles es que, entonces, sólo precisará una unidad Microdrive para servir a varios ordenadores).

El dispositivo de interconexión ('interface') RS232 es una conexión estandar que le permite conectar un Spectrum con teclados, impresoras, oredenadores y diversas otras máquinas aunque no estuvieran concebidas concretamente para el Spectrum.

Utilizarán algunas palabras clave suplementarias que están en el teclado, pero que no pueden emplearse sin los accesorios optativos y que son **OPEN #**, **CLOSE #**, **MOVE**, **ERASE**, **CAT** y **FORMAT**.

CAPITULO

23

IN y OUT

Resumen
OUT
IN

El procesador puede leer de, y (al menos con **RAM**) escribir en, la memoria utilizando **PEEK** y **POKE**. Al procesador, en sí mismo, no le importa realmente si la memoria es ROM, RAM o incluso nada en absoluto; sabe que hay 65536 direcciones de memoria y puede leer un byte de cada una (aunque sea algo sin sentido) y escribir un byte en cada una (aunque se pierda). De una forma completamente análoga, hay 65536 de los que se llaman "*ports*" de *E/S* (que significa conectores de entrada/salida). Estos se utilizan por el procesador para comunicar con elementos tales como el teclado o la impresora y pueden controlarse por el BASIC utilizando la función **IN** y la sentencia **OUT**.

IN es una función como **PEEK**.

IN dirección

Tiene un argumento, la dirección de «port», y su resultado es la lectura de un byte desde ese «port». **OUT** es una sentencia como **POKE**.

OUT dirección, valor

escribe el valor dado al «port» con la dirección dada. La forma en que se interpreta la dirección depende, en gran medida, del resto del ordenador; con bastante frecuencia, muchas direcciones diferentes significan lo mismo. En el Spectrum es mejor imaginar que la dirección se escribe en binario, porque los bits individuales tienden a operar con independencia. Hay 16 bits, que llamaremos (utilizando A para indicar dirección).

A15, A14, A13, A12, ..., A2, A1, A0

En este caso, A0 es el bit de peso 1, A1 es el bit de peso 2, A2 es el bit de peso 4 y as¡ sucesivamente, en correspondencia con el código binario. Los bits A0, A1, A2, A3 y A4 son los más importantes. Suelen ser «1», pero si alguno de ellos es 0, ello indica al ordenador que ha de realizar algo concreto. El ordenador no puede atender a más de una cosa a la vez, por lo que no más de uno de estos cinco bits debe ser 0. Los bits A5, A6 y A7 son ignorados, de modo que si usted es un experto en electrónica los puede emplear por su propia cuenta. Las mejores direcciones a utilizar son las que son un múltiplo de 32 menos 1, de modo que A0, ..., A4 son todos «1». Los bits A8, A9, etc., son utilizados, en ocasiones, para proporcionar información suplementaria.

El byte leído, o escrito, tiene 8 bits y se les suele denominar (utilizando D por datos como D7, D6, ..., D1, D0, se da una lista de las direcciones de 'ports' utilizadas.

Hay un conjunto de direcciones de entrada que leen el teclado y también la conexión de auricular EAR.

El teclado está dividido en 8 semifilas de 5 teclas cada una.

IN 65278 lee la semifila **CAPS SHIFT** a **V**
IN 65022 lee la semifila **A** a **G**
IN 64510 lee la semifila **Q** a **T**
IN 63486 lee la semifila **1** a **5**
IN 61438 lee la semifila **0** a **6**
IN 57342 lee la semifila **P** a **7**
IN 49150 lee la semifila **ENTER** a **H**
IN 32766 lee la semifila **SPACE** a **B**

(Estas direcciones son 254+256*(255-2↑n) cuando *n* va de 0 a 7).

En el byte leído, los bits D0 a D4 significan las cinco teclas en la semifila dada (D0 para la tecla exterior, D4 para la más cercana a la parte central. El bit es 0 si la tecla está pulsada y es 1 si no lo está. D6 es el valor en la conexión EAR.

La dirección de 'port' 254 utilizada como salida excita el altavoz (D4) y la conexión microfónica MIC (D3) y también establece el color del contorno (D2, D1 y DO).

La dirección de 'port' 251 maneja la impresora, tanto en lectura como en escritura: la lectura averigua si la impresora está preparada para imprimir más datos y la escritura envía puntos a imprimir.

Las direcciones de 'port' 254, 247 y 239 se emplean para los dispositivos suplementarios mencionados en el capitulo 22.

Ejecute el programa siguiente:

```
10 FOR n =0 TO 7: REM semifila número
20 LET a=254+256*(255-2↑n)
30 PRINT AT 0,0; IN a: GO TO 30
```

y juegue pulsando teclas. Cuando se haya aburrido con cada semifila, pulse **BREAK** y luego, teclee:

NEXT n

CAPITULO 24

La memoria

Resumen
CLEAR

En las profundidades del ordenador, todo se almacena en octetos (bytes), como, por ejemplo, los números entre 0 y 255. Se puede creer que se ha almacenado el precio de la lana o la dirección del vendedor de fertilizantes, pero todo ha sido convertido a conjuntos de octetos y lo único que el ordenador ve son octetos.

Todo lugar donde se puede almacenar un octeto posee una dirección, que es un número comprendido entre 0 y FFFFh (de modo que cualquier dirección puede almacenarse como un par de octetos), de modo que se puede considerar la memoria como si fuera una larga fila de casillas numeradas, cada una de las cuales puede contener su octeto (byte). Sin embargo, no todas las casillas son iguales, pues en la máquina estándar de 16K RAM las casillas numeradas del 8000h al FFFFh (1) faltan al completo. Las casillas del 4000h al 7FFFh son casillas de RAM, lo que quiere decir que se puede levantar la tapa y cambiarle el contenido. Finalmente, aquellas numeradas del 0 al 3FFFh son casillas ROM, que están recubiertas' de cristal y que no pueden abrirse. Todo lo que se puede hacer es leer lo que se depositó alí cuando se fabricó el ordenador.

ROM	RAM	No utilizado	
0	4000h =16384	8000h =32768	FFFFh =65535

Para inspeccionar el contenido de una casilla, empleamos la función **PEEK:** su argumento es la dirección de la casilla y su resultado el contenido. Por ejemplo, el programa siguiente imprime los primeros los primeros 21 octetos de la ROM (y sus direcciones):

```
10 PRINT "Direcciones"; TAB 8; "Octeto"
20 FOR a=0 TO 20
30 PRINT a; TAB 8; PEEK a
40 NEXT a
```

Es probable que todos estos octetos carezcan de significado para el lector, pero la pastilla procesadora los entiende como instrucciones que le indican qué hacer.

Para variar el contenido de una casilla (si es de RAM), empleamos la sentencia **POKE** (introducir), cuya forma es

POKE dirección, nuevo contenido

en donde la «dirección» y el «nuevo contenido» representan expresiones numéricas. Por ejemplo, si se dice

POKE 31000,57

(1) Nota: La letra h significa que está en hexadecimal.

al octeto de la dirección 31000 se le da el nuevo valor 57. Escríbase

PRINT PEEK 31000

para probarlo. (Trate de ejercer la función **POKE** con otros valores para comprobar que no hay trampa). Los nuevos valores han de estar comprendidos entre -255 y + 255 y si el elegido es negativo. entonces, se le añade 256.

La función **POKE** otorga un poder «inmenso» sobre el ordenador siempre que se la sepa emplear y posibilidades destructoras enormes si no se sabe. Es muy fácil echar a perder un programa que costó horas de teclado, mediante el ejercicio de **POKE** con un valor incorrecto en una dirección equivocada. Afortunadamente, ello no producirá al ordenador ningún daño permanente.

Echaremos, ahora, un vistazo más detallado a la manera en que se usa la RAM, pero no se moleste en leerlo si no le interesa el tema.

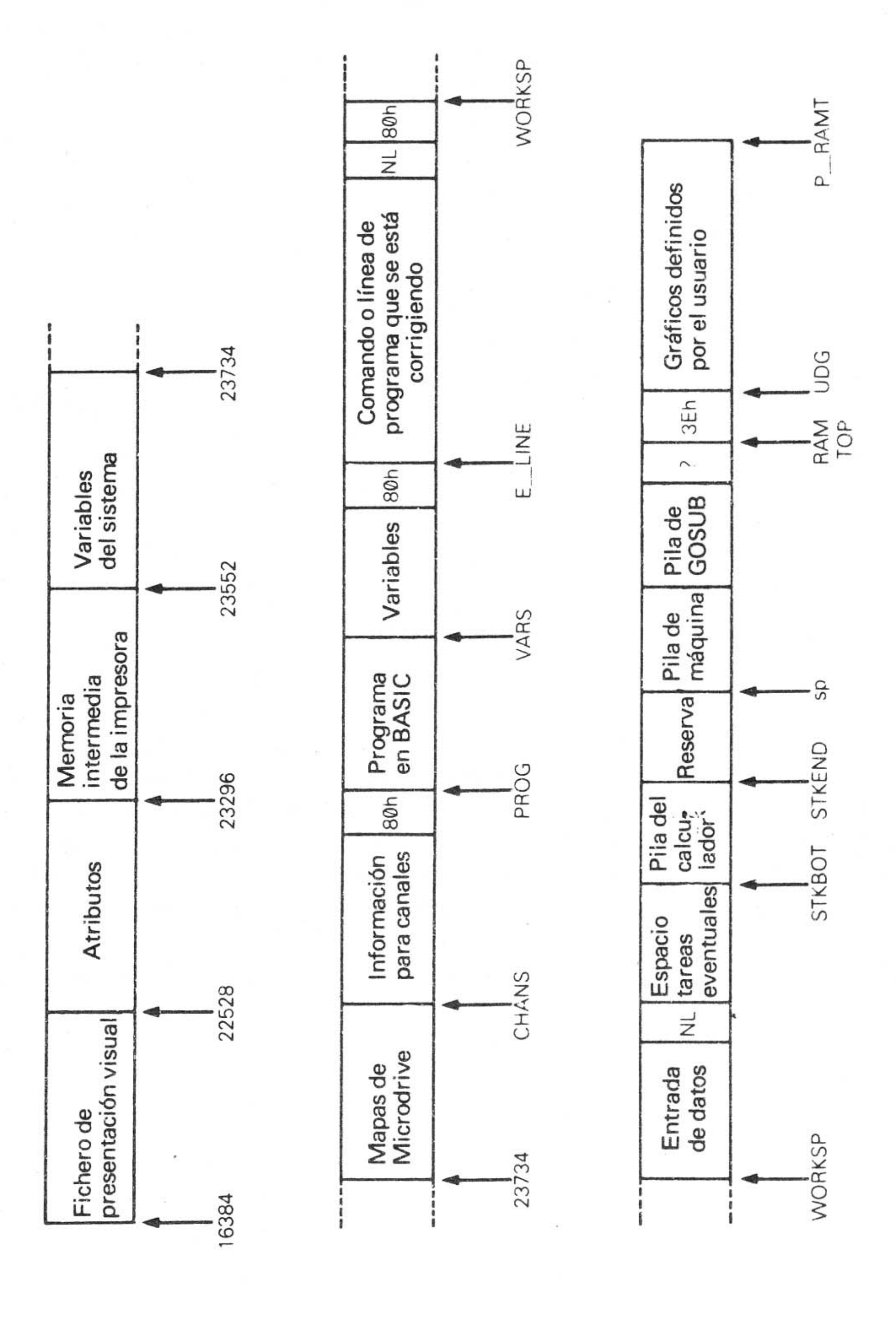
La memoria está dividida en zonas diferentes (que se muestran en el diagrama grande) para almacenar distintos tipos de información. Estas zonas son sólo suficientemente grandes para contener la información que contienen realmente, por lo que si se añade algo más en algún punto determinado (por ejemplo, incluyendo una línea de programa o una variable) se hace sitio empujando hacia arriba todo lo que queda por encima de este punto. Y al contrario si se borra información, entonces todo se empuja hacia abato.

El archivo para la representación en pantalla almacena una imagen de televisión. Resulta bastante curioso el modo en que está presentado, por lo que es probable que se prefiera no ejercer en ella las funciones **PEEK** y **POKE**. Cada carácter de la pantalla posee un cuadrado de 8x8 puntos y cada punto puede tomar el valor 0 (papel) o 1 (tinta). así, utilizando la notación binaria se puede almacenar la configuración como 8 octetos, uno para cada fila. Sin embargo, estos 8 octetos no se almacenan juntos, sino que lo hacen las filas correspondientes de los 32 caracteres de una línea sencilla, como una traza de 32 octetos, ya que esto es lo que necesita el haz electrónico de la televisión a medida que barre la pantalla de izquierda a derecha. Comoquiera que la imagen completa tiene 24 líneas de 8 barridos cada una, podría esperarse que hubiera que almacenar 172 barridos en total correlativos, pero no es verdad. Primero vienen las trazas superiores de las líneas 0 y 7, después, el siguiente barrido de las mismas líneas de 0 a 7 y así hasta llegar a la línea inferior' de las citadas líneas de 0 a 7. A continuación, se repite el mismo proceso para las líneas de 8 a 1 5 y, tras ello, se hace lo mismo con las líneas de 16 a 23. El resultado final de todo esto es que si estamos habituados a un ordenador que usa las funciones **PEEK** y **POKE** en la pantalla, habrá que cambiarse a usar **SCREEN$** y **PRINT AT** en su lugar, o **PLOT** y **POINT**.

Los atributos son los colores, etc., para cada posición de carácter, mediante el uso de **ATTR**. Todo ello se alma cena línea tras línea en el orden que se espera.

La memoria temporal de la impresora almacena los caracteres destinados a impresión.

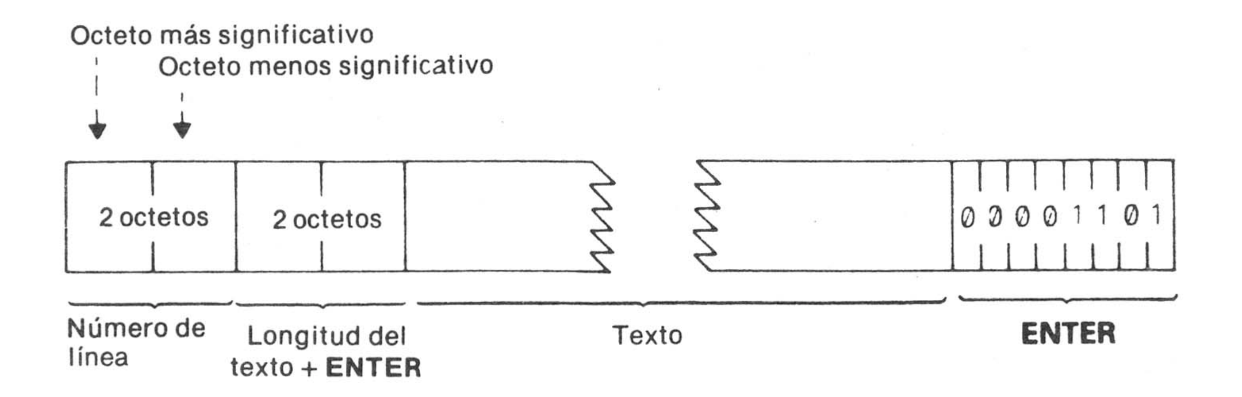
Las variables del sistema contienen varios elementos de información que indican al ordenador en qué estado se halla. Como están completamente listadas en el capítulo próximo. nos conformaremos por el momento con destacar que existen algunas (llamadas CHANS, PROG, VARS, E_LINE, etc.) que contienen las direciones de los límites entre !as diversas zonas de la memoria. No son variables de BASIC, por lo que sus nombres no serán identificados por el ordenador.


Fichero de presentación visual
Atributos
Memoria intermedia de la impresora
Variables del sistema
16384
22528
23296
23552
23734
Mapas de Microdrive
Información para canales
80h
Programa en BASIC
Variables
80h
Comando o línea de programa que se está corrigiendo
NL
80h
23734
CHANS
PROG
VARS
E_LINE
WORKSP
Entrada de datos
NL
Espacio tareas eventuales
Pila del calculador
Reserva
Pila de máquina
Pila de GOSUB
?
3Eh
Gráficos definidos por el usuario
WORKSP
STKBOT
STKEND
sp
RAM TOP
UDG
P_RAMT

Los mapas de la unidad Microdrive sólo existen cuando se utiliza este dispositivo, por lo que, normalmente, sus posiciones están vacías.

La zona para información de los canales contiene la información pertinente sobre los dispositivos de entrada y salida, es decir, el teclado (con la mitad inferior de la pantalla), la mitad superior de la pantalla y la impresora.

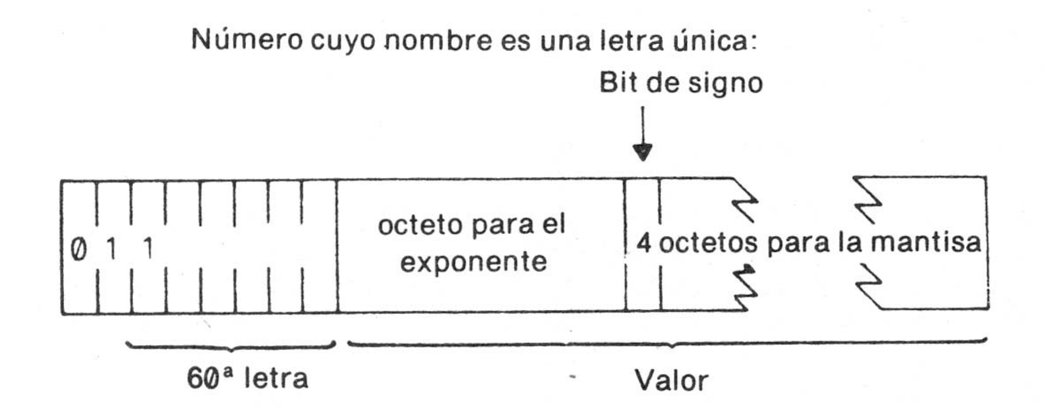
Cada línea del programa BASIC tiene la forma:



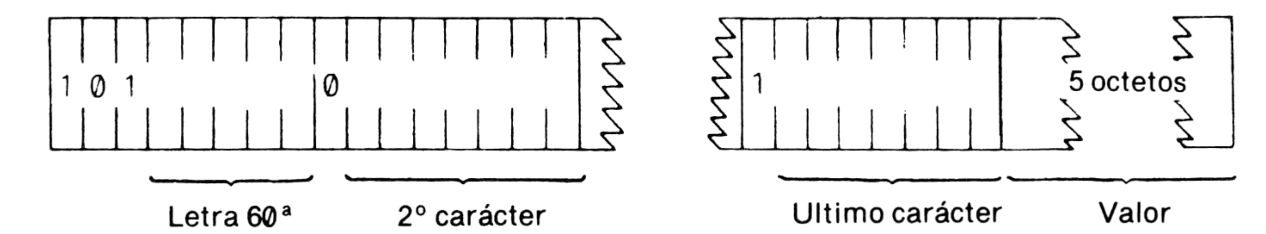
Nótese que, a diferencia con todos los demás casos de números representados con 2 bytes, en el Z80, el número de línea se almacenará aquí con el byte más significativo en cabeza: es decir: en el mismo orden en que se escriben.

Cualquier constante numérica del programa va seguida de su forma binaria, empleando el carácter **CHR$ 14** seguido de 5 bytes para el propio número.

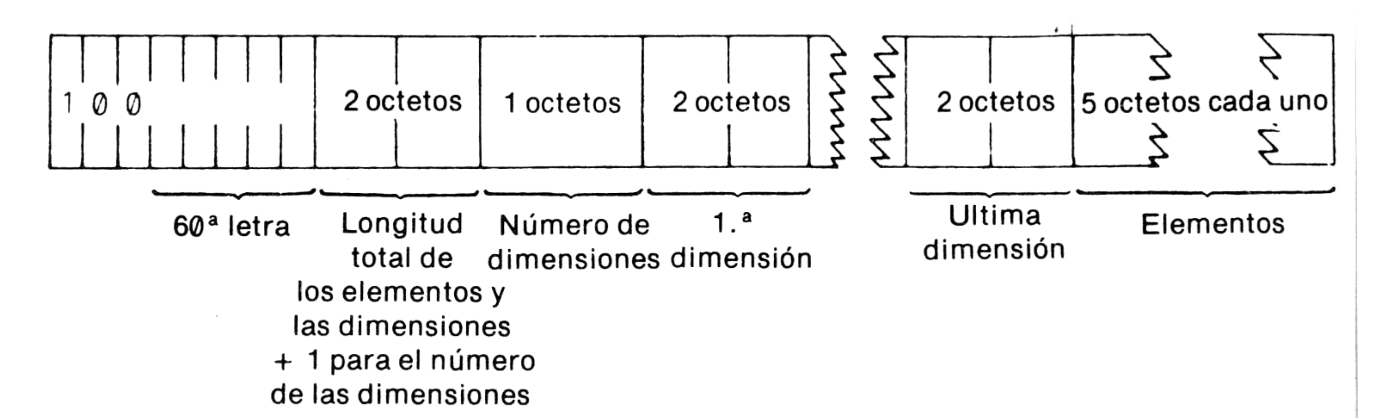
Las variables poseen formatos diferentes de acuerdo con sus distintas naturalezas. Las letras que componen los nombres deben ser imaginadas partiendo de las minúsculas:



Número cuyo nombre es más largo que una letra:



Matriz de números:

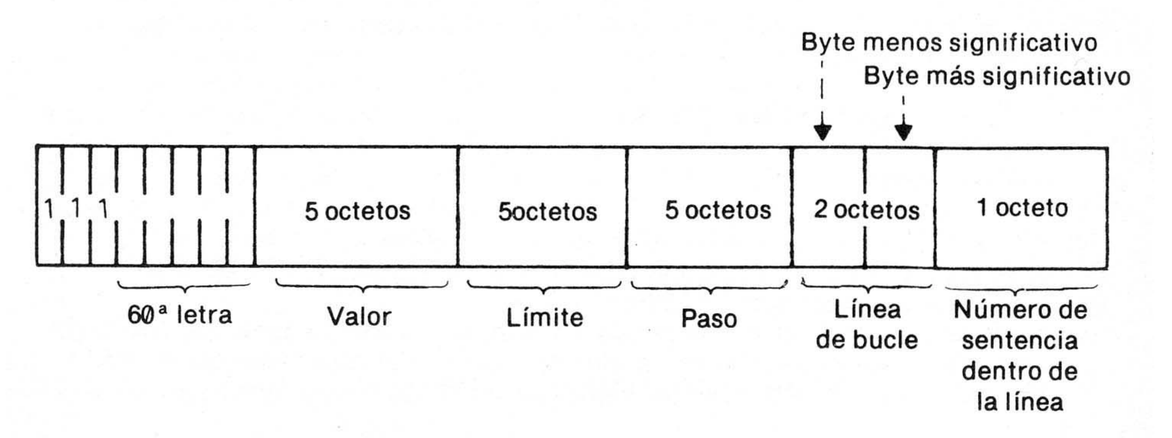


El orden de los elementos es:
primero, los elementos para los cuales el primer subíndice es 1, seguidamente, los elementos para los cuales el primer subíndice es 2
a continuación, los elementos para los cuales el primer subíndice es 3
y así sucesivamente para todos los valores posibles del primer subíndice.

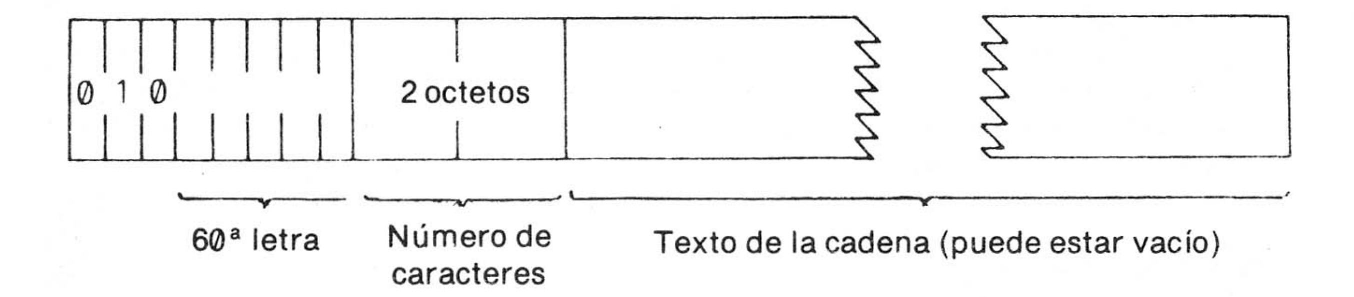
Los elementos con un primer subíndice dado se ordenan en la misma manera, empleando el segundo subíndice y así sucesivamente hasta alcanzar el último.

Como ejemplo, los elementos de la matriz b de 3*6 del capítulo 13 se almacenan en orden b(1,1) b(1,2) b(1,3) b(1,4) b(1,5) b(1,6) b(2,1) b(2,2) ... b(2,6) b(3,1) b(3,2) ... b(3,6).

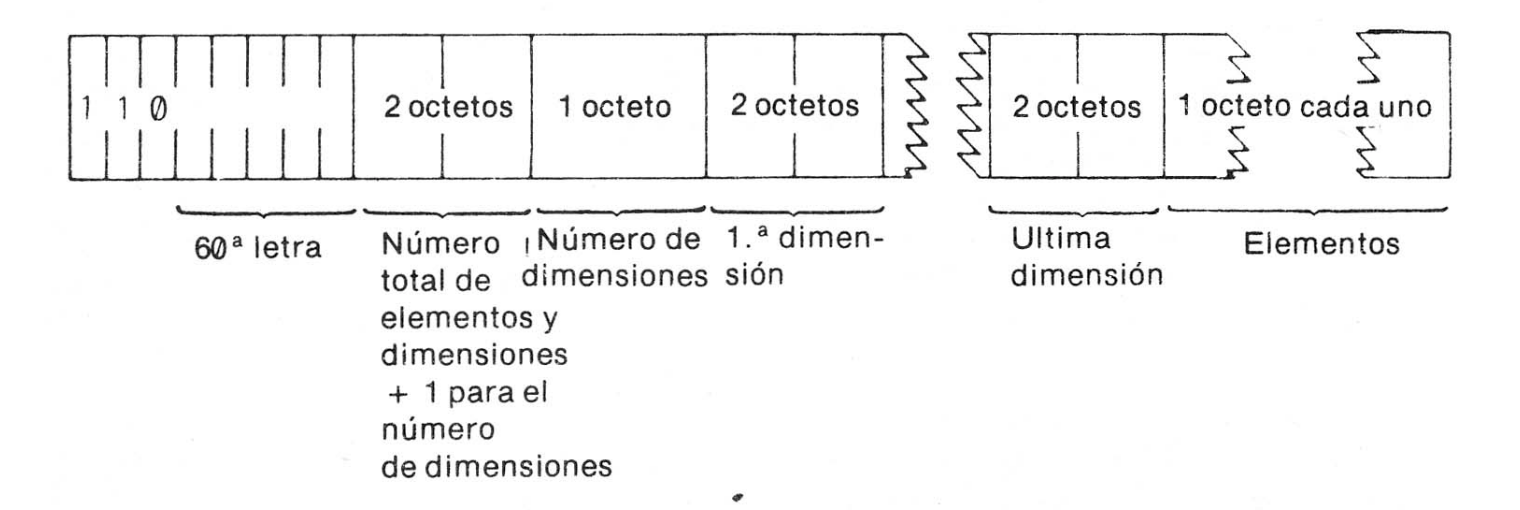
Variable de control para un bucle **FOR-NEXT**:



Cadenas



Matriz de caracteres:



El calculador es la parte del sistema BASIC que trata de la aritmética y los números con los que opera están contenidos en su mayoría en su pila de memoria.

La parte sobrante contiene el espacio no usado hasta ahora.

La pila de la máquina es la que usa el procesador del Z80 para contener las direcciones de retorno y elementos similares.

La pila para GOSUB se mencionó ya en el capítulo 5.

El byte hacia el que apunta RAMTOP tiene la d irección más alta que utiliza el sistema BASIC. Aún cuando **NEW** borra la RAM, lo hace sin alterar los gráficos definidos por el usuario. Se puede variar la dirección de RAMTOP poniendo un número en una sentencia **CLEAR**.

CLEAR nueva RAMTOP

Esta sentencia

(i) borra todas las variables
(ii) borra el fichero de la representación visual (como CLS)
(iii) reajusta la posición de PLOT a la esquina inferior izquierda.
(iv) ejecuta una restauración (RESTORE)
(v) borra la pila de GOSUB y la coloca en el nuevo límite definido por RAMTOP (suponiendo que queda entre la pila de cálculo y el final físico de la RAM). En cualquier otro caso, deja el límite del RAMTOP donde estaba.

RUN también produce borrado como **CLEAR**, aunque nunca altera el valor de la RAMTOP.

Utilizando de esta manera la instrucción **CLEAR** es posible tanto subir el límite RAMTOP de modo que se haga más sitio para el BASIC recubriendo los gráficos definidos por el usuario, como llevar el limite RAMTOP hacia abajo permitiendo más cantidad de RAM de la prevista en la instrucción **NEW**.

Teclee **NEW** y después **CLEAR 23800**, para hacerse una idea de lo que ocurre en la máquina cuando se llena.

Una de la primeras cosas que se notará cuando se comienza a escribir un programa es que el ordenador deja de aceptar entradas y hace sonar el zumbador. Esto significa que el ordenador está repleto y hay que vaciarlo ligeramente. Existen, además, dos mensajes de error cuyo significado es muy parecido: **4 Memory full** (Memoria completa) y **G No room for line** (No hay sitio para más líneas).

El zumbador suena también cuando se escribe una línea más larga de 23 líneas; sin embargo, el mensaje no se pierde, aunque no pueda verse (el zumbador suena como disuasión para escribir algo más).

Se puede ajustar la duración del zumbido mediante la instrucción **POKE** en la dirección 23608. La duración normal correspondiente al número 64 puede ser sustituida por otro número.

Cualquier número (a excepción del 0) puede escribirse únicamente como

± mx2e

donde ± es el signo

m es la *mantisa* y queda entre 1/2 y 1 (no puede ser 1),

y e es el *exponente*, un número entero (posiblemente negativo).

Supóngase que m se escribe en la escala binaria. Ya que es una fracción, poseerá un *punto binario* (como la coma, o punto, decimal en la escala de diez) seguido de una fracción binaria (igual que una fracción decimal); así, en binario

un medio se escribe .1
un cuarto se escribe .01
tres cuartos se escriben .11
una décima se escribe .000110011001100110011 ... etc. Nuestro número m, es menor que la unidad, por lo que su representación binaria carece de cifras antes del punto (coma en español) decimal. Y como quiera que su menor valor es 1/2, el bit que sigue inmediatamente al punto (coma en español) decimal es siempre un 1.

Para almacenar un número en el ordenador, disponemos de 5 octetos (bytes), de la manera que sigue:

(i) se escriben los ocho primeros bits de la mantisa en el segundo octeto (sabemos que el primer bit es un 1), los segundos ocho bits en el tercer octeto, los terceros ocho bits en el cuarto octeto y los cuartos ocho bits en el quinto octeto.

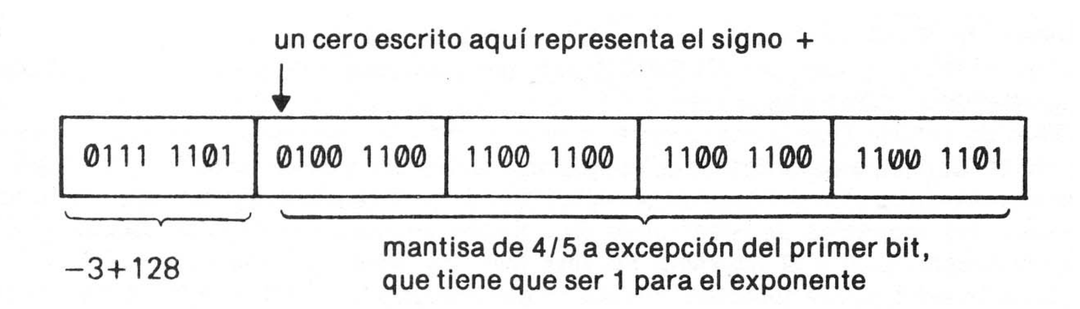
(ii) se sustituye el primer bit del segundo octeto (que sabemos que es un 1) por el signo: 0 para más y 1 para menos.

(iii) se escribe el exponente +128 en el primer octeto. Por ejemplo, supongamos que nuestro número es 1/10

1/10 = 4/5x2-3

Entonces, la mantisa m es 11001100110011001100110011001100 en binario (como

el bit 33° es un 1, redondearemos el bit 32° de 0 a 1) y el exponente e es -3. Aplicando las 3 reglas anteriores, encontramos los cinco octetos:



Existe un método alternativo para almacenar números enteros comprendidos entre -65535 y + 65535:

(i) el primer octeto es 0.
(ii) el segundo octeto es 0 para un número positivo y FFh para uno negativo.
(iii) los octetos tercero y cuarto son los octetos menos y más significativos, respec-
(iii) tivamente, del número (o el número +131072 si es negativo).
(iv) El quinto octeto (byte) es cero.

CAPITULO 25

Las variables del sistema

Los octetos de la memoria cuyas direcciones están comprendidas entre 235521 y 23733 se han reservado para usos específicos del sistema. Es posible examinarlas (**PEEK**) para averiguar carros detalles acerca del mismo y, algunas de ellas, se pueden también alterar (**POKE**). Las hemos listado aquí, junto con sus usos.

Se las ha llamado variarles del sistema y poseen nombres, pero no deben ser confundidas con las variables que usa el BASIC. El ordenador no identifica sus nombres como pertenecientes a las variables de sistema, por lo que aquí las exponemos únicamente como mnemotecnias para los «humanos».

Las abreviaturas de la columna 1 tienen los siguientes significados:

X Las variables no deben ser alteradas (**POKE** pues, de lo contrario el sistema resultará quebrantado.

N Las alteraciones de la variable no producirán efectos perjudiciales.

El numero asignado en la columna 1 es el de dos octetos que contiene la variable. Cuando hay dos octetos (bytes), el primero es el menos significativo - al contrario de lo que se podría esperar. De este modo, para variar el valor **v** de los dos octetos de una variable ubicada en la dirección **n**, usamos las sentencias:

POKE n,v-256*INT (v/1256)
POKE n + 1,ÍNT (v/256)

y para inspeccionar (**PEEK**) un valor, se usa la expresión

PEEK n + 256*PEEK (n + 1)

Notas	Dirección	Nombre	Contenido
N8	23552	KSTATE	Se usa para leer del teclado.
N1	23560	LAST K	Almacena el valor de la tecla pulsada últimamente.
1	23561	REPDEL	Tiempo (en 1/50 de segundo - o en 1/60 en Norteamérica) que debe presionarse la tecla para que se repita. Su valor parte de 35, pero es posible alterarlo (POKE) con otros valores.
1	23562	REPPER	Retardo (en 1/50 de segundo - o en 1/60 de segundo en Norteamérica) entre repeticiones sucesivas de una tecla mantenida oprimida: inicialmente vale 5.
N2	23563	DEFADD	Dirección del argumento de una función definida por el usuario, si es que se está valorando alguna; en otro caso vale 0.
N1	23565	K DATA	Almacena el segundo octeto del control del color introducido por el teclado.
N2	23566	TVDATA	Almacena octetos de color, controles **AT** y **TAB** que van a la televisión.
X38	23568	STRMS	Direcciones de canales de comunicación. de entrada y salida.

Notas	Dirección	Nombre	Contenido
2	23606	CHARS	Dirección del conjunto de caracteres menos 256 (que comienza por un espcaio y lleva el signo de copyright). Se encuentra normalmente en la ROM, pero es posible disponer una RAM para hacer que la función CHARS se dirija a la misma.
1	23608	RASP	Duración del zumbador de alarma.
1	23609	PIP	Duración del chasquido del teclado.
1	23610	ERR NR	El código de informe menos 1. Se inicializa a 255 (para -1) por lo que si se examina la posición 23610 (PEEK 23610) se obtiene 255.
X1	23611	FLAGS	Varios indicadores para control del sistema BASIC
X1	23612	TV FLAG	Indicadores asociados con la televisión.
X2	23613	ERR SP	Dirección del elemento de la pila de la máquina que es usado como retorno de error.
N2	23615	LIST SP	Dirección de la posición de retorno tras un listado automático.
N1	23617	MODE	Especifica el cursor K, L, C, E o G.
2	23618	NEWPPC	Línea a la que hay que saltar.
1	23620	NSPPC	Número de sentencia en una línea a la que hay que saltar. Si se ejerce la función **POKE** primeramente a NEWPPC, y luego a NSPPC, se fuerza un salto a una sentencia especificada en una línea.
2	23621	PPC	Número de línea de la sentencia en curso.
1	23623	SUBPPC	Número dentro de la línea de la sentencia en curso.
1	23624	BORDCR	Color del contorno *8; contiene también los atributos que se suelen usar para la mitad inferior de la pantalla.
2	23625	E PPC	Número de la línea en curso (con el cursor del programa).
X2	23627	VARS	Dirección de las variables.
N2	23629	DEST	Dirección de la variable en la asignación.
X2	23631	CHANS	Dirección del canal de datos.
X2	23633	CURCHL	Dirección de la información que se destina en ese momento para entrada y salida.
X2	23635	PROG	Dirección del programa BASIC.
X2	23637	NXTLIN	Dirección de la siguiente línea del programa.
X2	23639	DATADD	Dirección de la terminación del ú ltimo elemento de datos.
X2	23641	E LINE	Dirección del comando que se está escribiendo en el teclado.
2	23643	K CUR	Dirección del cursor.
X2	23645	CH ADD	Dirección del siguiente carácter a interpretar: el carácter que sigue al argumento de **PEEK**, o al de **NEWLINE** al final de una sentencia **POKE**.

Notas	Dirección	Nombre	Contenido
2	23647	X PTR	Dirección del carácter que sigue al signo ? Dirección del espacio eventual de trabajo.
X2	23651	STKBOT	Dirección del fondo de la pila de cálculo.
X2	23653	STKEND	Dirección del comienzo del espacio de reserva.
N1	23655	BREG	Registro b del calculador.
N2	23656	MEM	Dirección de la zona utilizada para la memoria del calculador. (Normalmente MEMBOT, pero no siempre).
1	23658	FLAGS2	Más indicadores
X1	23659	DF SZ	Número de líneas (incluida una en blanco) de la mitad inferior de la pantalla.
2	23660	S TOP	El número de la línea superior del programa en los listados automáticos.
2	23662	OLDPPC	Número de línea a la que salta la sentencia **CONTINUE.**
1	23664	OSPCC	Número dentro de la línea de la sentencia a la que salta la instrucción **CONTINUE**.
N1	23665	FLAGX	Indicadores varios.
N2	23666	STRLEN	Longitud del destino tipo de cadena en la asignación.
N2	23668	T ADDR	Dirección del siguiente elemento en la tabla sintática (muy difícilmente útil).
2	23670	SEED	El origen para **RND**, es la variable que se fija mediante la función **RANDOMIZE**.
3	23672	FRAMES	3 octetos (el menos significativo en primer lugar) del contador de cuadros. Se incrementa cada 20 ms. Véase el capítulo 18.
2	23675	UDG	Dirección del primer grafico definido por el usuario. Se la puede cambiar, por ejemplo, para ahorrar espacio a costa de tener menos gráficos definidos por el usuario.
1	23677	COORDS	Coordenada x del último punto trazado.
1	23678		Coordenada y del último punto trazado.
1	23679	P POSN	Número de 33 columnas de la posición de la impresora.
1	23680	PR CC	Octeto menos significativo correspondiente a la dirección de la siguiente posición a imprimir en la función **PRINT** (en la memoria temporal de la impresora).
1	23681		No se usa.
2	23682	ECHO E	Número de las 33 columnas y número de las 24 líneas (en la mitad inferior) del final de la memoria temporal de entrada.
2	23684	DF CC	Dirección de la posición de **PRINT** en el fichero de la presentación visual.
2	23686	DFCCL	Igual que DFCC, pero para la mitad inferior de la pantalla.

Notas	Dirección	Nombre	Contenido
X1	23688	S POSN	Número de las 33 columnas de la posición de **PRINT**.
X1	23689		Número de las 24 líneas de la posición de **PRINT**.
X2	23690	SPOSNL	Igual que S POSN, pero para la mitad inferior.
1	23692	SCR CT	Contaje de los desplazamientos hacia arriba («scroll»): es siempre una unidad más que el número de desplazamiento que se van a hacer antes de terminar en un **scroll?** Si se cambia este valor con un número mayor que 1 (digamos 255), la pantalla continuará «enrollándose» sin necesidad de nuevas instrucciones.
1	23693	ATTR P	Colores permanentes en curso, etc. (tal y como los fijaron las sentencias para el color).
1	23694	MASK P	Se usa para los colores transparentes, etc. Cualquier bit que sea 1 indica que el bit correspondiente de los atributos no se toma de ATTR P, sino de lo que ya está en pantalla.
N1	23695	ATTR T	Colores en uso temporal, etc. (tal como los fijaron los elementos de color).
N1	23696	MASK T	Igual que MASK P, pero temporal.
1	23697	P FLAG	Más indicadores
N30	23698	MEMBOT	Zona de memoria destinada al calculador; se usa para almacenar números que no pueden ser guardados convenientemente en la pila del calculador.
2	23728		No se usa.
2	23730	RAMTOP	Dirección del último octeto del BASIC en la zona del sistema.
2	23732	P-RAMT	Dirección del último octeto de la RAM física.

Este programa proporciona los primeros 22 octetos de la zona de variables:

```
10 FOR n=0 TO 21
20 PRINT PEEK (PEEK 16400+256*PEEK 16401 +n)
30 NEXT n
```

Trate de encajar la variable de control **n** con las descripciones anteriores. A continuación, sustitúyase la línea 20 por

```
20 PRINT PEEK (16509 + n)
```

Esto último proporciona los primeros 22 octetos de la zona del programa. Compáreselo con el propio programa.

CAPITULO 26

Empleo del código máquina

Resumen
USR con argumento numérico

Este capítulo está concebido para quienes entienden el código máquina del Z80, que es el conjunto de instrucciones que utiliza el microprocesador Z80. Si no es su caso, pero le gustaría conocerlo, hay mucha bibliografía sobre esta materia y no le será difícil encontrar un libro adecuado en una librería especializada.

Estos programas suelen estar escritos en **lenguaje ensamblador**, que, aunque sea algo especial, no resulta demasiado difícil de entender con la práctica (puede consultar las instrucciones de lenguaje ensamblador en el apéndice A). Sin embargo, para ejecutarlos en el ordenador necesita codificar el programa en una secuencia de bytes (en esta forma, se denomina código máquina). Esta traducción suele hacerse por el propio ordenador, con el empleo de un programa llamado **ensamblador**. No hay ningún ensamblador incorporado en el Spectrum, pero puede adquirir uno en cassette. Si no lo adquiere, tendrá que hacer la traducción por sí mismo, a condición de que el programa no sea demasiado largo.

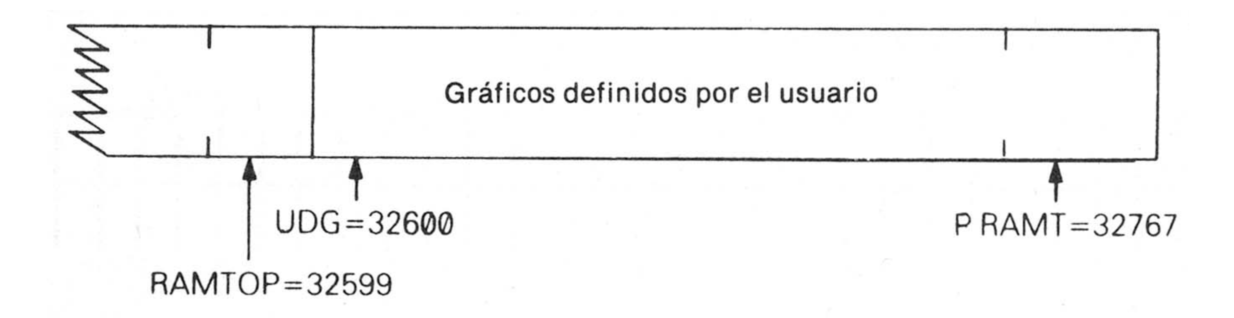
A título de ejemplo, sea el programa:

```
1d, bc, 99
ret
```

que carga el par de registros bc con 99. Traduce a los cuatro bytes de código máquina 1,99,0 (por Id, bc, 99) y 201 (por ret). (Si busca 1 y 201 en el Apéndice A, encontrará Id, bc, NN (en donde NN significa cualquier número de dos bytes) y ret.).

Cuando tenga su programa de código máquina, el siguiente paso es conseguir su introducción en el ordenador (probablemente, un ensamblador haría esta tarea de forma automática). Necesita decidirse en qué lugar de la memoria ha de ponerse y lo mejor que se puede hacer es conseguir espacio suplementario para tal objeto entre la zona del BASIC y los gráficos definidos por el usuario.

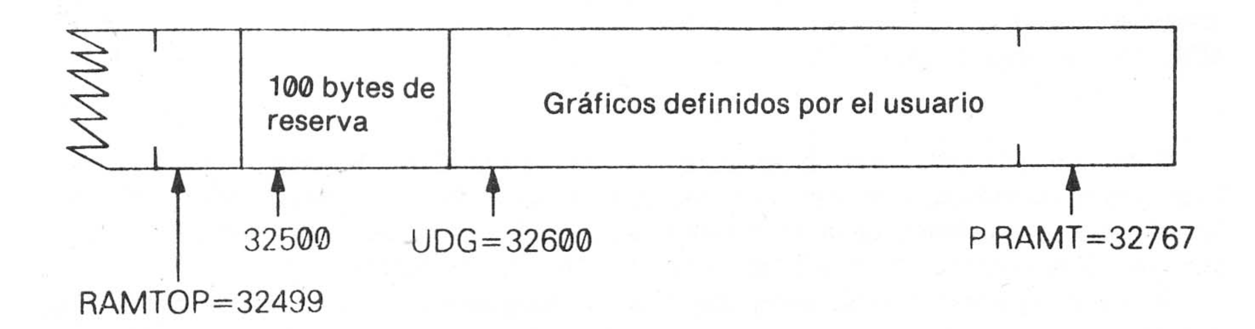
Supongamos, por ejemplo, que tiene una memoria de 16K para el Spectrum. Para comenzar, el extremo superior de la RAM tiene:



Si teclea:

CLEAR 32499

le proporcionará un espacio de 100 (para estar seguro) bytes que comienza en la dirección 32500.



Para introducir el programa en código máquina, ha de ejecutar un programa en BASIC como el siguiente:

```
10 LET a = 32500
20 READ n: POKE a,n
30 LET a = a + 1: GO TO 20
40 DATA 1,99,0,201
```

(se producirá una interrupción con el informe **E Out of DATA** cuando haya rellenado los cuatro bytes que especificó).

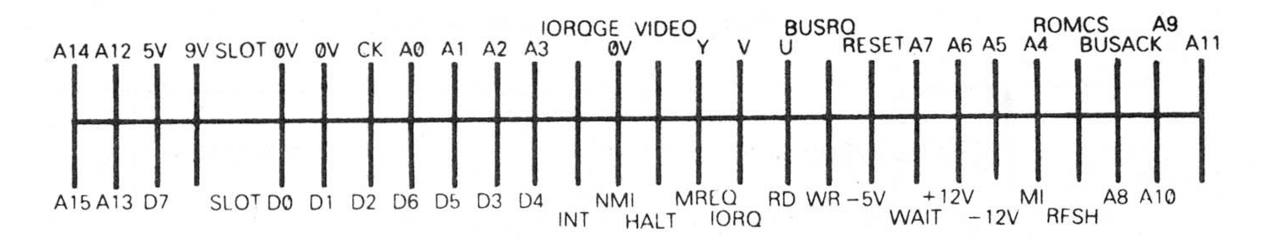
Para ejecutar el código máquina, ha de emplear la función **USR**, pero esta vez con un argumento numérico, la dirección de comienzo. Su resultado es el valor del registro bc en el retorno desde el programa en código máquina, por consiguiente, si hace:

```
PRINT USR 32500
```

obtendrá la respuesta 99.

La dirección de retorno al BASIC se almacena en la forma habitual, por lo que el retorno se hace mediante una instrucción ret del Z80. No debe emplear los registros iy e i en una rutina en código máquina.

Los buses de control, de datos y de direcciones están accesibles en la parte posterior del Spectrum, por lo que con un Spectrum puede hacer casi todo lo que pudiera realizar con un Z80. A veces, sin embargo, al hardware del Spectrum podría plantear inconvenientes. Damos la ilustración de un diagrama de las conexiones expuestas en la parte posterior.



Puede conservar su programa en código máquina bastante fácilmente con:

SAVE "algún nombre" **CODE 32500,4**

Por lo que se ha visto, no hay ninguna forma de realizar la conservación de modo que cuando se cargue automáticamente se ejecute por sí mismo, pero puede eludir esta dificultad con el empleo de un programa en BASIC.

```
10 LOAD " " CODE 32500,4
20 PRINT USR 32500
```

Haga primero:

SAVE "algún nombre" **LINE**

y luego:

SAVE "xxxx" CODE 32500,4
LOAD "algún nombre"

que cargará y ejecutará automáticamente el programa en BASIC y este último cargará y ejecutará el código máquina.

El Juego de Caracteres

A continuación se da el juego de caracteres del Spectrum completo, con códigos en decimal y hexadecimal. Si se imagina los códigos como las instrucciones de código máquina del Z80, entonces, las columnas de la derecha dan los mnemotécnicos (nemónicos en la jerga informática) de lenguaje ensamblador correspondientes. Como probablemente sepa si está versado en estos temas, algunas instrucciones del Z80 son compuestas comenzando con CBh o EDh; las dos columnas de la derecha las indican.

Código	*Carácter*	*Hex*	*Ensamblador del Z80*	*-Después de CB*	*-Después de ED*
0		00	nop	rlc b	
1		01	ld bc, NN	rlc c	
2		02	ld (bc), a	rlc d	
3		03	inc bc	rlc e	
4		04	inc b	rlc h	
5		05	dec b	rlc l	
6	**PRINT** coma	06	ld b,N	rlc (hl)	
7	**EDIT**	07	rlca	rlc a	
8	cursor izquierda	08	ex af,af’	rrc b	
9	cursor derecha	09	add hl,bc	rrc c	
10	cursor abajo	0A	ld a,(bc)	rrc d	
11	cursor arriba	0B	dec bc	rrc e	
12	**DELETE**	0C	inc c	rrc h	
13	**ENTER**	0D	dec c	rrc l	
14	número	0E	ld c,N	rrc (hl)	
15	no utilizado	0F	rrca	rrc a	
16	**INK** control	10	djnz DIS	rl b	
17	**PAPER** control	11	ld de,NN	rl c	
18	**FLASH** control	12	ld (de),a	rl d	
19	**BRIGHT** control	13	inc de	rl e	
20	**INVERSE** control	14	inc d	rl h	
21	**OVER** control	15	dec d	rl l	
22	**AT** control	16	ld d,N	rl (hl)	
23	**TAB** control	17	rla	rl a	
24		18	jr DIS	rr b	
25		19	add hl, de	rr c	
26		1A	ld a,(de)	rr d	
27		1B	dec de	rr e	
28		1C	inc e	rr h	
29		1D	dec e	rr l	
30		1E	ld e,N	rr (hl)	
31		1F	rra	rr a	
32	espacio	20	jr nz,DIS	rr (hl)	
33	!	21	ld hl,NN	sla c	
34	“	22	ld (NN),hl	sla d	
35	#	23	inc hl	sla e	
36	$	24	inc h	sla h	
37	%	25	dec h	sla l	

Código	*Carácter*	*Hex*	*Ensamblador del Z80*	*-Después de CB*	*-Después de ED*
38	&	26	ld h,N	sla (hl)	
39	‘	27	daa	sla a	
40	(	28	jr z,DIS	sra b	
41	)	29	add hl,hl	sra c	
42	*	2A	ld hl,(NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc l	sra h	
45	-	2D	dec l	sra l	
46	.	2E	ld l,N	sra (hl)	
47	/	2F	cpl	sra a	
48	0	30	jr nc,DIS		
49	1	31	ld sp,NN		
50	2	32	ld (NN),a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	ld (hl),N		
55	7	37	scf		
56	8	38	jr c,DIS	srl b	
57	9	39	add hl,sp	srl c	
58	:	3A	ld a,(NN)	srl d	
59	;	3B	dec sp	srl e	
60	<	3C	inc a	srl h	
61	=	3D	dec a	srl l	
62	>	3E	ld a,N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	40	ld b,b	bit 0,b	in b,(c)
65	A	41	ld b,c	bit 0,c	out (c),b
66	B	42	ld b,d	bit 0,d	sbc hl,bc
67	C	43	ld b,e	bit 0,e	ld (NN),bc
68	D	44	ld b,h	bit 0,h	neg
69	E	45	ld b,l	bit 0,l	retn
70	F	46	ld b,(hl)	bit 0,(hl)	im 0
71	G	47	ld b,a	bit 0,a	ld i,a
72	H	48	ld c,b	bit 1,b	in c,(c)
73	I	49	ld c,c	bit 1,c	out (c),c
74	J	4A	ld c,d	bit 1,d	adc hl,bc
75	K	4B	ld c,e	bit 1,e	ld bc,(NN)
76	L	4C	ld c,h	bit 1,h	
77	M	4D	ld c,l	bit 1,l	reti
78	N	4E	ld c,(hl)	bit 1,(hl)	
79	O	4F	ld c,a	bit 1,a	ld r,a
80	P	50	ld d,b	bit 2, b	in d,(c)
81	Q	51	ld d,c	bit 2,c	out (c),d

Código	*Carácter*	*Hex*	*Ensamblador del Z80*	*-Después de CB*	*-Después de ED*
82	R	52	ld d,d	bit 2, d	sbc hl,de
83	S	53	ld d,e	bit 2,e	ld (NN), de
84	T	54	ld d,h	bit 2,h	
85	U	55	ld d,l	bit 2,l	
86	V	56	ld d,(hl)	bit 2,(hl)	im 1
87	W	57	ld d,a	bit 2,a	ld a,i
88	X	58	ld e,b	bit 3,b	in e,(c)
89	Y	59	ld e,c	bit 3,c	out (c),e
90	Z	5A	ld e,d	bit 3,d	adc hl,de
91	[	5B	ld e,e	bit 3,e	ld de,(NN)
92	/	5C	ld e,h	bit 3,h	
93	]	5D	ld e,l	bit 3,l	
94	↑	5E	ld e,(hl)	bit 3,(hl)	im 2
95	___	5F	ld e,a	bit 3,a	ld a,r
96	£	60	ld h,b	bit 4,b	in h,(c)
97	a	61	ld h,c	bit 4,c	out (c),h
98	b	62	ld h,d	bit 4,d	sdc hl,hl
99	c	63	ld h,e	bit 4,e	ld (NN),hl
100	d	64	ld h,h	bit 4,h	
101	e	65	ld h,l	bit 4,l	
102	f	66	ld h,(hl)	bit 4,(hl)	
103	g	67	ld h,a	bit 4,a	rrd
104	h	68	ld l,b	bit 5,b	in l,(c)
105	i	69	ld l,c	bit 5,c	out (c),l
106	j	6A	ld l,d	bit 5,d	adc hl,hl
107	k	6B	ld l,e	bit 5,e	ld hl,(NN)
108	l	6C	ld l,h	bit 5,h	
109	m	6D	ld l,l	bit 5,l	
110	n	6E	ld l,(hl)	bit 5,(hl)	
111	o	6F	ld l,a	bit 5,a	rld
112	p	70	ld (hl),b	bit 6,b	in f,(c)
113	q	71	ld (hl),c	bit 6,c	
114	r	72	ld (hl),d	bit 6,d	sbc hl,sp
115	s	73	ld (hl),e	bit 6,e	ld (NN),sp
116	t	74	ld (hl),h	bit 6,h	
117	u	75	ld (hl),l	bit 6,l	
118	v	76	halt	bit 6,(hl)	
119	w	77	ld (hl),a	bit 6,a	
120	x	78	ld a,b	bit 7,b	in a,(c)
121	y	79	ld a,c	bit 7,c	out (c),a
122	z	7A	ld a,d	bit 7,d	adc hl,sp
123	{	7B	ld a,e	bit 7,e	ld sp,(NN)
124	|	7C	ld a,h	bit 7,h	
125	}	7D	ld a,l	bit 7,l	

Código	*Carácter*	*Hex*	*Ensamblador del Z80*	*-Después de CB*	*-Después de ED*
126	~	7E	ld a,(hl)	bit 7,(hl)	
127	©	7F	ld a,a	bit 7,a	
128		80	add a,b	res 0,b	
129		81	add a,c	res 0,c	
130		82	add a,d	res 0,d	
131		83	add a,e	res 0,e	
132		84	add a,h	res 0,h	
133		85	add a,l	res 0,l	
134		86	add a,(hl)	res 0,(hl)	
135		87	add a,a	res 0,a	
136		88	adc a,b	res 1,b	
137		89	adc a,c	res 1,c	
138		8A	adc a,d	res 1,d	
139		8B	adc a,e	res 1,e	
140		8C	adc a,h	res 1,h	
141		8D	adc a,l	res 1,l	
142		8E	adc a,(hl)	res 1,(hl)	
143		8F	adc a,a	res 1,a	
144	(a)	90	sub b	res 2,b	
145	(b)	91	sub c	res 2,c	
146	(c)	92	sub d	res 2,d	
147	(d)	93	sub e	res 2,e	
148	(e)	94	sub h	res 2,h	
149	(f)	95	sub l	res 2,l	
150	(g)	96	sub (hl)	res 2,(hl)	
151	(h)	97	sub a	res 2,a	
152	(i)	98	sbc a,b	res 3,b	
153	(j)	99	sbc a,c	res 3,c	
154	(k)	9A	sbc a,d	res 3,d	
155	(l)	9B	sbc a,e	res 3,e	
156	(m)	9C	sbc a,h	res 3,h	
157	(n)	9D	sbc a,l	res 3,l	
158	(o)	9E	sbc a,(hl)	res 3,(hl)	
159	(p)	9F	sbc a,a	res 3,a	
160	(q)	A0	and b	res 4,b	ldi
161	(r)	A1	and c	res 4,c	cpi
162	(s)	A2	and d	res 4,d	ini
163	(t)	A3	and e	res 4,e	outi
164	(u)	A4	and h	res 4,h	
165	**RND**	A5	and l	res 4,l	
166	**INKEY**$	A6	and (hl)	res 4,(hl)	
167	**PI**	A7	and a	res 4,a	
168	**FN**	A8	xor b	res 5,b	ldd
169	**POINT**	A9	xor c	res 5,c	cpd

Código	*Carácter*	*Hex*	*Ensamblador del Z80*	*-Después de CB*	*-Después de ED*
170	**SCREEN$**	AA	xor d	res 5,d	ind
171	**ATTR**	AB	xor e	res 5,e	outd
172	**AT**	AC	xor h	res 5,h	
173	**TAB**	AD	xor l	res 5,l	
174	**VAL$**	AE	xor (hl)	res 5,(hl)	
175	**CODE**	AF	xor a	res 5,a	
176	**VAL**	B0	or b	res 6,b	ldir
177	**LEN**	B1	or c	res 6,c	cpir
178	**SIN**	B2	or d	res 6,d	inir
179	**COS**	B3	or e	res 6,e	otir
180	**TAN**	B4	or h	res 6,h	
181	**ASN**	B5	or l	res 6,l	
182	**ACS**	B6	or (hl)	res 6, (hl)	
183	**ATN**	B7	or a	res 6,a	
184	**LN**	B8	cp b	res 7,b	lddr
185	**EXP**	B9	cp c	res 7,c	cpdr
186	**INT**	BA	cp d	res 7,d	indr
187	**SQR**	BB	cp e	res 7,e	otdr
188	**SGN**	BC	cp h	res 7,h	
189	**ABS**	BD	cp l	res 7,l	
190	**PEEK**	BE	cp (hl)	res 7,(hl)	
191	**IN**	BF	cp a	res 7,a	
192	**USR**	C0	ret nz	set 0,b	
193	**STR$**	C1	pop bc	set 0,c	
194	**CHR$**	C2	jp nz,NN	set 0,d	
195	**NOT**	C3	jp NN	set 0,e	
196	**BIN**	C4	call nz,NN	set 0,h	
197	**OR**	C5	push bc	set 0,l	
198	**AND**	C6	add a,N	set 0,(hl)	
199	**<=**	C7	rst 0	set 0,a	
200	**>=**	C8	ret z	set 1,b	
201	**<>**	C9	ret	set 1,c	
202	**LINE**	CA	jp z,NN	set 1,d	
203	**THEN**	CB		set 1,e	
204	**TO**	CC	call z,NN	set 1,h	
205	**STEP**	CD	call NN	set 1,l	
206	**DEF FN**	CE	adc a,N	set 1,(hl)	
207	**CAT**	CF	rst 8	set 1,a	
208	**FORMAT**	D0	ret nc	set 2,b	
209	**MOVE**	D1	pop de	set 2,c	
210	**ERASE**	D2	jp nc,NN	set 2,d	
211	**OPEN #**	D3	out (N),a	set 2,e	
212	**CLOSE #**	D4	call nc,NN	set 2,h	
213	**MERGE**	D5	push de	set 2,l	

Código	*Carácter*	*Hex*	*Ensamblador del Z80*	*-Después de CB*	*-Después de ED*
214	**VERIFY**	D6	sub N	set 2,(hl)	
215	**BEEP**	D7	rst 16	set 2,a	
216	**CIRCLE**	D8	ret c	set 3,b	
217	**INK**	D9	exx	set 3,c	
218	**PAPER**	DA	jp c,NN	set 3,d	
219	**FLASH**	DB	in a,(N)	set 3,e	
220	**BRIGHT**	DC	call c,NN	set 3,h	
221	**INVERSE**	DD	prefijos instruc- ciones empleando ix	set 3,l	
222	**OVER**	DE	sbc a,N	set 3,(hl)	
223	**OUT**	DF	rst 24	set 3,a	
224	**LPRINT**	E0	ret po	set 4,b	
225	**LLIST**	E1	pop hl	set 4,c	
226	**STOP**	E2	jp po,NN	set 4,d	
227	**READ**	E3	ex (sp),hl	set 4,e	
228	**DATA**	E4	call po,NN	set 4,h	
229	**RESTORE**	E5	push hl	set 4,l	
230	**NEW**	E6	and N	set 4,(hl)	
231	**BORDER**	E7	rst 32	set 4,a	
232	**CONTINUE**	E8	ret pe	set 5,b	
233	**DIM**	E9	jp (hl)	set 5,c	
234	**REM**	EA	jp pe,NN	set 5,d	
235	**FOR**	EB	ex de,hl	set 5,e	
236	**GO TO**	EC	call pe,NN	set 5,h	
237	**GO SUB**	ED		set 5,l	
238	**INPUT**	EE	xor N	set 5,(hl)	
239	**LOAD**	EF	rst 40	set 5,a	
240	**LIST**	F0	ret p	set 6,b	
241	**LET**	F1	pop af	set 6,c	
242	**PAUSE**	F2	jp p,NN	set 6,d	
243	**NEXT**	F3	di	set 6,e	
244	**POKE**	F4	call p,NN	set 6,h	
245	**PRINT**	F5	push af	set 6,l	
246	**PLOT**	F6	or N	set 6,(hl)	
247	**RUN**	F7	rst 48	set 6,a	
248	**SAVE**	F8	ret m	set 7,b	
249	**RANDOMIZE**	F9	ld sp,hl	set 7,c	
250	**IF**	FA	jp m,NN	set 7,d	
251	**CLS**	FB	ei	set 7,e	
252	**DRAW**	FC	call m,NN	set 7,h	
253	**CLR**	FD	prefijos instruc- ciones empleando iy	set 7,l	
254	**RETURN**	FE	cp N	set 7,(hl)	
255	**COPY**	FF	rst 56	set 7,a	

Informes

Aparecen en las dos últimas líneas de la pantalla cada vez que el ordenador cesa de ejecutar el programa BASIC y explican la causa de su parada, sea por razones naturales, sea porque se ha producido algún error.

El informe posee un número o letra codificados que hace posible acudir a la tabla que a continuación se presenta, en donde un breve mensaje explica lo que ha ocurrido, así como el número de línea y de sentencia dentro de la línea donde se detuvo el programa. (Como línea 0 aparece un comando. Dentro de una línea, la sentencia 1 es la del principio, la sentencia 2 sigue inmediatamente al primer signo de dos puntos o a **THEN**, y así sucesivamente).

El comportamiento de **CONTINUE** depende, en gran manera, de los mensajes de información. Normalmente, **CONTINUE** va a la línea y sentencia especificadas en el último informe, pero hay excepciones con los informes 0, 9 y D. (Véase también el Apéndice C).

He aquí la tabla que muestra todos los informes. También indica las circunstancias en que puede producirse el mensaje y ello hace referencia al Apéndice C. Por ejemplo, el error **A Invalid argument** (Argumento no válido), puede producirse con **SQR, IN, ACS** y **ASN**, siendo las entradas de estas funciones, en el apéndice C, las encargadas de señalar exactamente qué argumentos no son válidos.

Código	*Significado*	*Situación*
0	OK Terminación satisfactoria o salto a un número de línea mayor que todos los existentes. Este informe no altera la línea y sentencia a que salta **CONTINUE**	Cualquiera
1	NEXT without FOR (Next sin For) No existe variable de control (no fue fijada por una secuencia FOR), pero existe una variable ordinaria con el mismo nombre.	**NEX**
2	Variable not found (Variable no hallada) Para el caso de una variable sencilla, ocurrirá siempre que haya sido asignada para introducir una sentencia **LET**, **READ** o **INPUT**, cargada desde la cinta magnética, o fijada por una sentencia **FOR**. En el caso de una variable con subíndice, ocurrirá si se la usa antes de que haya sido definida mediante una sentencia **DIM** o cargada de la cinta magnética.	Cualquiera
3	Subscript wrong (subíndice erróneo) Un subíndice queda fuera de las dimensiones de la matriz, o hay un número equivocado de subíndice. Si es negativo, o mayor que 65535; entonces aparecerá el código 13 de error.	Variables con subíndices y subcadenas

Código	*Significado*	*Situación*
4	Out of memory (Fuera de memoria) No hay bastante sitio en el ordenador, para lo que se pretende hacer. Si el ordenador parece realmente haberse bloqueado en este estado, habrá que anular la linea de comando (u órdenes) usando la sentencia DELETE y después anular una línea de programa, o quizá dos (con la intención de restablecerlas después) para permitir la utilización de, por ejemplo, CLEAR.	**LET, INPUT, FOR, DIM, GO SUB, LOAD, MERGE**, a veces durante la valoración las expresiones
5	Out of screen (Fuera de pantalla) Una sentencia **INPUT** ha pretendido generar más de 23 líneas en la mitad inferior de la pantalla. Ocu-rre también con **PRINT AT** 22...	**INPUT, PRINT AT**
6	Number too big (Número demasiado grande) El cálculo ha llevado a la máquina a un número mayor que 1038 aproximadamente.	Cualquier operación aritmética
7	RETURN without GO SUB (Retorno sin GO SUB) Ha habido una sentencia **RETURN** más que las **GO SUB** encontradas.	**RETURN**
8	End of file (Final de fichero)	Operaciones de microdrive, etc.
9	STOP statemente (Sentencia Stop) Tras ella, la sentencia **CONTINUE** no repetirá el **STOP**, sino lleva a cabo la sentencia siguiente.	**STOP** (alto)
A	Invalid Argument (Argumento erróneo) El argumento de una función no es correcto por alguna razón.	**SQR, LN, ASN, ACS, USR** (con argumento en cadena)
B	Integer out of range (Entero fura de margen) Cuando se requiere un entero, el argumento del punto (la coma en español) flotante, se redondea al entero más próximo. Si ello queda fuera de un margen conveniente, entonces, aparece el error B. Para acceso a las matrices, véase también el código n° 3 de errores.	**RUN, RANDOMIZE, POKE, DIM, GO TO, GO SUB, LIST, LLIST, PAUSE, PLOT, CHR$, PEEK, USR** (con argumento numérico). Acceso a matrices.
C	Nonsense in BASIC (sin significado en BASIC) El argumento (cadena) no constituye una expresión válida.	**VAL, VAL$**
D	BREAK - CONT repeats (se repite BREAK durante alguna operación con los periféricos. El comportamiento de CONTINUE, después de este informe, es normal en el sentido que repite la sentencia. Compáreselo con el informe L.	**LOAD, SAVE, VERIFY, MERGE LPRINT, LLIST, COPY** También cuando el ordenador pregunta **scroll?** y se le responde con **N, SPACE** o **STOP**
E	Out of DATA (No quedan datos) Se ha tratado de leer (**READ**) más allá del final de la	

Código	*Significado*	*Situación*
	de la lista de datos.	
F	Invalid file name (Nombre de archivo no válido) Se ha asignado a la función **SAVE** un nombre vacío o de más de 10 caracteres.	**SAVE**
G	No hay bastante sitio en la memoria para admitir una nueva línea del programa.	Cuando se añade una línea al programa
H	STOP in INPUT (STOP en los datos de entrada) Algún dato de entrada (**INPUT**) empezaba con **STP**, o se apretó esta tecla durante la introducción de una línea (**INPUT LINE**). Al contrario que el informe 9, tras el informe H la función **CONTINUE** se comportará normalmente si se repite la sentencia **INPUT**	**INPUT**
I	For without NEXT (FOR sin NEXT) Había un lazo **FOR** para ser ejecutado ninguna vez (p.e. **FOR** n=1 **TO 0**) y la sentencia **NEXT** correspondiente no aparece.	**FOR**
J	Invalid I/O device (Dispositivo de entrada/salida no válida)	Operaciones con Microdrive
K	Invalid colour (Color no válido) El número especificado no es un valor apropiado.	**INK, PAPER, BORDER, FLASH, BRIGHT, INVERSE, OVER**, también tras uno de los caracteres de control correspondientes
L	BREAK into Program (Hay un BREAK en el programa. La tecla de BREAK fue oprimida y ello se detectó entre dos sentencias. El número de línea y el de sentencia que aparecen en el informe se refieren a la sentencia en curso, antes de que se oprimiera la tecla de **BREAK**, pero la sentencia **CONTINUE** salta a la siguiente (permitiendo que se ejecute cualquier salto), por lo que no se repite ninguna sentencia.	Cualquiera
M	RAMTOP no good (RAMTOP incorrecto) El número especificado como RAMTOP es muy grande o muy pequeño.	**CLEAR,** posiblemente en **RUN**
N	Statement lost (sentencia perdida). Salta a una sentencia que ya no existe.	**RETURN, NEXT,** y **CONTINUE**
O	Invalid Stream (Serie no válida)	Operaciones con el microdrive, etc.
P	FN without DEF (FN sin DEF) Función definida por el usuario.	**FN**

Código	*Significado*	*Situación*
Q	Parameter error (parámetro erróneo) Número equivocado de argumentos, o en uno de ellos es de tipo equivocado (cadena en lugar de número o viceversa).	**FN**
R	Tape loading error (error de carga en la cinta) Se ha encontrado un fichero en la cinta magnética que no puede ser leído por alguna razón, o que no se verifica.	**VERIFY LOAD** o **MERGE**

Una descripción del ZX Spectrum para referencia

El primer apartado de este apéndice es una repetición de la parte de la introducción que se ocupa del teclado y la pantalla.

El teclado

Los caracteres del ZX Spectrum no sólo comprenden los *símbolos* sencillos (letras, dígitos, etc.), sino también los comandos (palabras clave, nombres de función, etc.) y todos ellos se introducen por teclado sin tener que ser deletreados. Para conseguir todas estas funciones y comandos u órdenes algunas teclas poseen 5, y más, significados distintos, dados en parte mediante el cambio de teclas (es decir, oprimiendo la tecla **CAPS SHIFT** o la **SYMBOL SHIFT** a la vez que la tecla considerada) y en parte, por tener la máquina en los diferentes *modos*.

El modo está indicado por el cursor , una letra que parpadea e indica dónde se va a insertar el siguiente carácter del teclado.

El modo K (Keyword -palabra clave) reemplaza automáticamente al modo L cuando la máquina está esperando una orden o una línea del programa (en vez de datos de entrada, o **INPUT** data) y de su posición en la línea se sabe lo que debe venir, si un número de línea o una palabra clave. Esto es al principio de la línea, o exactamente detrás de **THEN**, o exactamente detrás del símbolo «**:**» (excepto en una cadena). Si no se cambia de modo, la siguiente pulsación de tecla se interpretará bien como una palabra clave (escrita sobre la tecla) o bien como un dígito.

El modo L (letras) es el modo en el que por defecto, de encuentra el ordenador una vez introducido un comando. En este modo, se puede escribir cualquier carácter alfanumérico.

En ambos modos K y L, una tecla pulsada a la vez que la de **SYMBOL SHIFT** se interpreta como el carácter rojo escrito sobre la tecla. Y un número pulsado a la vez que la tecla de **CAPS SHIFT** se interpreta como la función de control escrita en blanco sobre la tecla. La tecla de **CAPS SHIFT**, en modo K, no produce ningún efecto y en el modo L convierte las minúsculas en mayúsculas.

El modo C (de mayúsculas, o CAPITALS) es una variante del modo L en el que todas las letras aparecen en mayúsculas. **CAPS LOCK** es una tecla que produce el cambio del modo L al modo C y al revés.

El modo E (extendido) se usa para obtener otros comandos («tokens»). Se accede pulsando simultáneamente las dos teclas de **SHIFT** (**CAPS SHIFT** y **SIMBOL SHIFT**) y permanece activo durante una pulsación.

Una vez que haya obtenido el cursor **E**, podrá escribir cualquier comando indicado en verde (parte superior de cada tecla), simplemente pulsando la tecla deseada. Para escribir aquellos comandos que estén señalados en rojo (parte inferior de cada tecla) -y los que también se accede en modo **E** -, deberá pulsar las dos teclas **SHIFT**, y a continuación, pulsar la tecla **SYMBOL SHIFT** juntamente con la tecla deseada.

El modo G (GRAPHICS) se activa cuando se pulsa la tecla **GRAPHICS** (**CAPS SHIFT** y **9**) y permanece hata que se vuelve a pulsar dicha tecla. Una tecla numérica produce un gráfico tipo mosaico, a excepción de las teclas **GRAPHICS** (9) y **DELETE** (0) que desavtivan el modo gráfico o borran un carácter respectivamente. El resto de las teclas, excepto las teclas v, w, x, y y z, producen un gráfico definido por el usuario.

Cualquier tecla que sea retenida por más de 2 ó 3 segundos produce la repetición automática.

Los símbolos introducidos por teclado aparecen en la parte inferior de la pantalla y se insertan en la posición en la que se encuentra el cursor. El cursor se puede desplazar hacia la izquierda con **CAPS SHIFT** y **5**, o hacia la derecha con **CAPS SHIFT** y **8**. Se puede borrar el carácter anterior al cursor con **DELETE** (**CAPS SHIFT** y **0**). (Nota: se puede borrar la línea entera tecleando **EDIT** (**CAPS SHIFT** y **1**) seguido de **ENTER**).

Cuando se pulsa **ENTER**, se ejecuta la línea, se la incluye en el programa, o se la toma como dato de entrada, según convenga salvo que contenga algún error de sintaxis. En este caso aparecerá junto al error un **?** intermitente.

A medida que se introducen líneas de programa se va presentando su listado en la mitad superior de la pantalla. La forma en que se origina la lista es bastante complicada y se explica más completamente en el capítulo 2. La última línea introducida se denomina línea en curso y está indicada por el símbolo **>** , pero se la puede cambiar con la ayuda de las teclas ↑ (**CAPS SHIFT** y **6**) y ↓ (**CAPS SHIFT** y **7**).

Si se pulsa **EDIT** (**CAPS SHIFT** y **1**), la línea en curso se copia en la mitad inferior de la pantalla y puede ser corregida por edición.

Cuando se ejecuta una orden, o se pone en marcha un programa, la salida se presenta en la mitad superior de la pantalla y queda allí hasta que se introduce una línea de programa, se pulsa **ENTER** junto con una línea vacía o se pulsa ↑ o ↓. En la mitad inferior aparece un informe que muestra un código (dígito o letra) referenciado en el Apéndice B, y que significa el número de la línea que contiene la última sentencia ejecutada (o 0 si es una orden) y la posición de la sentencia dentro de la línea. El informe permanece en la pantalla hasta que se pulsa alguna tecla (que cambia al modo K).

En ciertas circunstancias, las teclas **CAPS SHIFT** y **SPACE** actúan en conjunto como un **BREAK** (ruptura) parando el ordenador, apareciendo el informe D o el L. Esto se reconoce:

(i) al final de una sentencia, cuando hay un programa en marcha,
(ii) cuando el ordenador está usando el cassette o la impresora.

La pantalla de televisión

Tiene 24 líneas de 32 caracteres cada una y está dividida en dos partes. La superior ocupa como máximo 22 líneas y tiene, bien un listado, o bien la salida que entrega un programa. Cuando la escritura, en la parte superior, alcanza su límite inferior, el desplazamiento («scrolling») de una línea; si ello implica la pérdida de una línea que aún no ha podido ser examinada, entonces el ordenador se detiene con el mensaje **scroll?** Al pulsar las teclas **N**, **SPACE** o **STOP**, el programa se para y uestra el informe **D BREAK-CONT repeats**. Si se pulsa cualquier otra tecla, se permite el «scrolling». La parte inferior de la pantalla se utiliza para introducir los comandos, líneas de programas, y los datos de entrada (**INPUT** data) y también para mostrar los informes. La parte inferior empieza con 2 líneas (de las cuales la superior está vacía), pero puede ampliarse para admitir cualquier información que se introduzca por el teclado.

Cuando la posición del cursor alcanza la mitad superior, las ampliaciones sucesivas producirán el «scrolling» de esta parte de la pantalla.

La posición de cada carácter posee atributos que especifican el color de su papel

(fondo) y el de su tinta (primer plano), en dos niveles de brillo y si debe destellar o no. Los colores posibles son negro, azul, rojo, violeta, verde, amarillo y blanco.

El borde de la pantalla puede fijarse de uno de estos colores empleando el comando **BORDER**.

La posición de un carácter está formada por una matriz de 8x8 elementos (pixels) y los gráficos de alta resolución se obtienen fijando bien la tinta o el papel de cada elemento individual de la matriz.

Las propiedades de la posición de un carácter se ajustan según se escriba un carácter o se trace un elemento individual (pixel). La forma exacta del ajuste está determinada por los parámetros de impresión, de los que hay dos juegos (denominados permanente y temporal) de seis: los **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE** y **OVER**. Los parámetros permanentes de la parte superior están fijados por las sentencias **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE** y **OVER** y duran hasta nueva disposición. (Inicialmente se componen de tinta negra sobre papel blanco, con brillo normal, sin destello, video normal y sin reimpresión). Los parámetros permanentes de la parte inferior utilizan el color del borde como color del papel, con tinta de contraste de color blanco o negro, brillo normal, sin destello, video normal y sin reimpresión.

Los parámetros temporales están fijados por los elementos **PAPER**, **INK**, etc., que están contenidos en las sentencias **PRINT**, **LPRINT**, **INPUT**, **PLOT**, **DRAM** y **CIRCLE** y también por los caracteres de control **PAPER**, **INK**, etc., cuando se escriben en la pantalla van seguidos de un octeto adicional para especificar el valor del parámetro. Los parámetros temporales duran sólo hasta el final de la sentencia **PRINT** (o lo que sea), o en las sentencias de entrada (**INPUT**), hasta que se necesitan datos de entrada (**INPUT** data) del teclado, siendo, entonces, sustituidos por los parámetros permanentes.

Los parámetros **PAPER** e **INK** varían de 0 a 9. Los parámetros de 0 a 7 son los colores que se usan cuando se representa el carácter:

0 negro
1 azul
2 rojo
3 violeta
4 verde
5 azul verdoso (cyan)
6 amarillo
7 blanco

El parámetro 8 («transparencia») especifica que el color de la pantalla ha de ser conservado al imprimir el carácter.

El parámetro 9 («contraste») especifica que el color en cuestión (del papel o de la tinta) va a ser bie blanco o negro, para resaltar con el otro color.

Los parámetros **FLASH** destello) y **BRIGHT** (brillo) son 0, 1 ó 8: el significado de 1 es que se activa el destello o el brillo, 0, que se desactiva y 8 (transparente) que no se altera en ninguna posición.

Los parámetros **OVER** e **INVERSE** son 0 ó 1.

OVER 0 los caracteres nuevos borran los antiguos

OVER 1 las configuraciones de bits de ambos caracteres nuevo y antiguo se combinan en una operación **OR** exclusiva (sobreimpresión)

INVERSE 0 los caracteres nuevos se escriben en el color de la tinta sobre el color del papel (video normal)

INVERSE 1 los caracteres nuevos se escriben en el color del papel sobre el color de la tinta (videro invertido).

Cuando la televisión recibe el carácter de control **TAB** (tabulador) espera dos nuevos octetos que especifiquen el punto n de parada (el octeto menos significativo en primer lugar). Este es convertivo a módulo 32, (llamémosle no) y después se dejan espacios suficientes para desplazar la posición del cursor a la columna no.

Cuando se recibe el carácter de control coma, se dejan espacios suficientes (uno al menos) para llevar el cursor a la columna 0 o a la columna 16.

Cuando se recibe el carácter de control **ENTER** se lleva el cursor a la línea siguiente, manteniendo la misma posición.

La impresora

La salida de información a la impresora del ZX se efectúa utilizando una memoria temporal de una línea de 32 caracteres de longitud y se envía una línea a la impresora:

(i) cuando se completa una línea
(ii) cuando se recibe un carácter **ENTER**
(iii) al final del programa, si queda algo que imprimir.
(iv) cuando un carácter de control, **TAB** o coma, lleva la posición de impresión a una nueva línea.

Los controles de **TAB** y coma producen espacios de la misma forma en la televisión.

El control **AT** cambia la posición de impresión usando el número de columna e ignora el número de línea.

La impresora es sensible a los comandos **INVERSE** y **OVER** (y también a las sentencias correspondientes) de la misma manera que la pantalla, pero no a los de **PAPER**, **INK**, **FLASH** o **BRIGHT**.

La impresora se para mostrando un error B si se pulsa **BREAK**.

Si la impresora no está conectada, la salida hacia ella se pierde simplemente.

El BASIC

Los números se almacenan con una exactitud de 9 ó de 10 dígitos . El mayor número que puede conseguir es 1038 aproximadamente y el número (positivo) más pequeño es $4*10^{-39}$

Un número se almacena en el ZX Spectrum en binario de coma flotante con un byte de exponente e (1 < = e < = 255) y cuatro bytes de mantisa m (1/2 < = m < 1). Ello representa el número $m*2^{e-128}$

Puesto que 1/2 < = m < 1, el bit más significativo de la mantisa m es siempre 1. Por consiguiente, en la realidad, podemos sustituirlo con un bit para indicar el signo (0 para números positivos y 1 para negativos).

Los números enteros pequeños tienen una representación especial en la que el primer byte es 0, el segundo es un bit de signo (O o FFh) y los bytes tercero y cuarto son los enteros en la forma de complemento a 2, con el byte menos significativo primero.

Las variables numéricas tienen nombres de longitud arbitraria, que comienzan con una letra y continúan con letras y dígitos. Los espacios y los controles de colores se ignoran y todas las letras se convierten en minúsculas.

Las variables de control de los bucles **FORT-NEXT** tienen nombres con una sola letra de longitud.

Las matrices numericas tienen nombres de una sola letra de longitud, que pueden ser los mismos que el nombre de una variable simple. Pueden tener, arbitrariamente, muchas dimensiones de magnitud arbitraria. Los subíndice comienzan en 1.

Las cadenas tienen una longitud completamente flexible. El nombre de una cadena está constituido por una sola letra seguida por $.

Las matrices de cadena pueden tener, arbitrariamente, muchas dimensiones de magnitud arbitraria. El nombre es una sola letra seguida por $ y quizás no sea el mismo que el nombre de una cadena. Todas las cadenas en una matriz dada tienen la misma longitud fija, que se especifica como una dimensión final suplementaria en la sentencia **DIM**. Los subíndices comienzan en 1.

Fragmentación («troceado»): Las subcadenas de cadenas pueden especificarse con el empleo de «slicers» (troceadores), que pueden ser

(i) vacío

o

(ii) expresión numérica

o

(iii) expresión numérica optativa **TO** expresión numérica optativa y se emplea para expresar una subcadena mediante

(a) una expresión de cadena («slicer»)
(b) variable de matriz de cadena (subíndice..., subíndice, 'slicer')

que significa lo mismo que

variable de matriz de cadena (subíndice, .. , subíndice) ('slicer').

En (a), se supone que la expresión de cadena tiene el valor s$.

Si el 'slicer' está vacío, el resultado es s$ considerado como una subcadena de sí mismo.

Si es una expresión numérica con valor m, entonces, el resultado es el m-ésimo carácter de s$ (una subcadena de longitud 1).

Si tiene la forma (iii), entonces se supone que la primera expresión numérica tiene el valor m (el valor por defecto es 1) y el segundo, n (el valor por defecto es la longitud de s$).

Si 1 < = m < = n < = longitud de s$, entonces, el resultado es la subcadena de s$ que comienza con el m-ésimo carácter y que finaliza con el n-ésimo.

Si 0 < = n < m, entonces, el resultado es la cadena vacía.

De cualquier otro modo, se produce error 3.

La fragmentación se realiza antes de que se evalúen las funciones u operaciones, a no ser que los paréntesis determinen otra cosa.

Subcadenas pueden ser objeto de asignación (ver LET).

Si unas comillas de cadena han de escribirse en un literal de cadena, entonces, deben hacerse dobles.

Funciones

El argumento de una función no necesita paréntesis si es una constante o una variable (con subíndice o «troceada», posiblemente).

Función	*Tipo de argumento* x	*Resultado*
ABS	número	Magnitud absoluta.
ACS	número	Arco coseno en radianes. Error A si x no está en el margen de -1 a +1
AND	operación binaria el operando de la derecha es siempre un número. Operando de la izquierda numérico: Operando de la izquierda es una cadena:	$A \textbf{ AND } B = \begin{cases} A \text{ si } B <> 0 \\ A\$ \text{ si } B = 0 \end{cases}$ $A\$ \textbf{ AND } B = \begin{cases} 0 \text{ si } B <> 0 \\ A\$ \text{ si } B = 0 \end{cases}$
ASN	número	Arco seno en radianes. Error A si x no está en el margen de -1 a +1
ATN	número	Arco tangente en radianes.
ATTR	dos argumentos, x e y, ambos números; encerrados entre paréntesis	Un número cuya forma binaria codifica los atributos de línea x y columna y en la televisión. El bit 7 (más significativo) es 1 para parpadeo y 0 para no parpadeo. El bit 6 es para brillo especial y 0 para brillo normal. Los bits 5 a 3 son el color del papel. Los bits 2 a 0 son el color de tinta. Error B a menos que 0 < x <= 23 y 0 <= y <= 31
BIN		No se trata realmente de una función, sino una notación alternativa para números: BIN seguida

Función	*Tipo de argumento*	*Resultado*
		por una secuencia de «0» y de «1» es el número con dicha representación en binario.
CHR$	número	El carácter cuyo código es x, redondeado al entero más próximo.
CODE	cadena	El código del primer carácter en x (o 0 si x es la cadena vacía).
COS	número (en radianes)	Coseno x
EXP	número	e^x
FN		**FN** seguida por una letra llama una función definida por el usuario (ver **DEF**). los argumentos deben estar encerrados entre paréntesis; aunque no haya argumentos deben seguir existiendo los paréntesis.
IN	número	Se produce una lectura de datos a nivel de procesador desde el ‘port’ x (0 <= x <= FFFF) (carga el par de registros bc con x y la instrucción en lenguaje ensamblador en a(c)).
INKEY$	ninguno	Efectúa la lectura del teclado. El resultado es el carácter que representa (en el modo L o C) la tecla pulsada si hay exactamente una y si hay más, la cadena vacía.
INT	número	Parte entera (siempre se redondea por defecto).
LEN	cadena	Longitud
LN	número	Logaritmo natural (con base e) Error A si x <= 0
NOT	número	0 si x<>0, 1 si x=0. **NOT** tiene prioridad 4.
OR	operación binaria, ambos operandos son números	a **OR** b = { 1 si b<> 0 ; a si b= 0 a **OR** tiene prioridad 2
PEEK	número	El valor del byte en memoria cuya dirección es x (redondeando al entero más próximo) Error B si x no está en el margen de 0 a 65535)
PI	ninguno	π (3,14159265 ...)
POINT	Dos argumentos, x e y, ambos números; encerrados entre paréntesis	1 si el pixel en (x,y) es color de la tinta, 0 si es el color del papel. Error B a menos que 0 <= x <= 255 y 0 <= y <= 175
RND	ninguno	El siguiente número pseudoaletorio en una secuencia generada tomando la potencias de 75 módulo 65537,restando 1 y dividiendo por 65536. 0 <= y < 1
SCREEN$	dos argumentos, x e y, ambos números; encerrados entre paréntesis	El carácter que aparece, bien sea normalmente bien sea invertido, en la pantalla de televisión en la línea x y en la columna y. Proporciona la cadena vacía, si no se identifica el carácter.

Función	*Tipo de argumento*	*Resultado*
		Error B a menos que 0 <= x <= 23 y 0 <= y <= 31
SGN	número	Signo: el signo de x (-1 para negativo, 0 para 0, 0+1 para positivo)
SIN	número (en radianes)	Seno x
SQR	número	Raíz cuadrada Error A si x < 0
STR$	número	La cadena de caracteres que se visualizaría si se hubiera impreso x.
TAN	número (en radianes)	Tangente
USR	número	Llama la subrutina de código máquina cuya dirección de comienzo es x. En el retorno de ejecución, el resultado es el contenido del par de registros bc.
USR$	cadena	La dirección de la configuración de bits para el gráfico definido por el usuario correspondiente a x. Error A si x no es una sola letra entre a y u o un gráfico definido por el usuario.
VAL	cadena	Evalúa x (sin sus comillas de límite) con una expresión numérica.
VAL$	cadena	Error C si x contiene un error de sintaxis o da un valor de cadena. Otros errores posibles, como para **VAL**.
-	número	Negación

Lo que sigue son operaciones binarias:

+	Adición (en números) o concatenación (en cadenas) Resta
-	Resta
*	Multiplicación
/	División
↑	Elevación a una potencia. Error B si el operando de la izquierda es negativo
=	igual a
>	mayor que
<	menor que
<=	menor que o igual a
>=	mayor que o igual a
<>	distinto

Ambos operandos deben ser del mismo tipo. El resultado es un número 1, si la comparación se mantiene y 0 si no es así. distinto a Operaciones binarias:

Las funciones y las operaciones tienen las prioridades siguientes:

Operación	**Prioridad**
Asignación de subíndices y fragmentación	12
Todas las funciones salvo NOT y operación unaria 'menos'	11
$\uparrow$ Potenciación	10
Operación unaria 'menos' (es decir, 'menos' solo usado para negar algo)	9
*, /	8
+ , - ('menos' usado para restar un número de otro)	6
=, >, <, <=, >=, <>	5
NOT	4
AND	3
OR	2

Sentencias

En esta lista

α	representa una sola letra
v	representa una variable
x. y, z,	representan expresiones numéricas
m, n	representan expresiones numéricas que se redondean al entero más próximo
e	representa una expresión
f	representa una expresión valorada de cadena
s	representa una secuencia de sentencias separadas por dos puntos :
c	representa una secuencia de elementos de color, cada uno terminado por comas o punto y coma ;. Un elemento de color tiene la forma de una sentencia **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE** u **OVER**.

Observe que se permiten expresiones arbitrarias en cualquier lugar (salvo para el número de línea al principio de una sentencia).

Todas ls sentencias, salvo **INPUT**, **DEF** y **DATA**, pueden utilizarse como comandos o en programas (aunque sean más sensibles en un caso que en el otro). Una línea de programa o comando pueden tener varias sentencias, separadas por dos puntos (:). No hay ninguna restricción sobre el lugar de una línea en donde pueda producirse cualquier sentencia particular (aunque vea **IF** y **REM**).

BEEP x, y

Suena una nota a través del altavoz durante x segundos a una altura de y semitonos por encima de la nota «do» media (o tono menor, si y es negativo).

BORDER m

Establece el color del contorno de la pantalla y también el color del papel para la parte inferior de la pantalla.

Error K si m no está en el margen de 0 a 7

BRIGHT	Establece el brillo de los caracteres posteriormente impresos. n=0 para brillo normal, 1 para brillo especial y 8 para transparente.
CAT	No opera sin Microdrive, etc.
CIRCLE x,y,z	Dibuja un arco de circunferencia, de centro (x,y) y radio z.
CLEAR	Borra todas las variables, liberando el espacio que ocupaban. Efectúan **RESTORE** y **CLS**, repone la posición **PLOT** a la esquina inferior izquierda y borra la pila de subrutinas **GO SUB**.
CLEAR n	Como **CLEAR**, pero, de ser posible, cambia la variable del sistema **RAMTOP** a n y pone allí la nueva pila de **GO SUB**.
CLOSE #	No opera sin Microdrive, etc.
CLS	(Borra pantalla). Borra el fichero de presentación visual.
CONTINUE	Continúa el programa, comenzando en donde se dejó la última vez que se interrumpió con informe distinto a 0. Si el informe fuera 9 o L, entonces, continúa con la siguiente sentencia (teniendo en cuenta los saltos); de no ser así, repite aquélla en donde se produjo el error. Si el último informe estaba en una línea de comando, entonces. **CONTINUE** intentará continuar la línea de comando y pasará a un bucle si el error estaba en 0:1, dará informe 0 si estaba en 0:2 o da un error **N** si estaba en 0:3 o mayor. **CONTINUE** aparece como **CONT** en el teclado.
COPY	Envía una reproducción del contenido de la pantalla a la impresora, si está conectada; de no ser así, se queda sin hacer nada. Observe que **COPY** no puede emplearse para imprimir los listados automáticos que aparecen en la pantalla. Informe D si se pulsa BREAK
DATA e_1, e_2, e_3, ..	Parte de la lista de **DATA**. Debe estar incluida en el programa.
DEF FN $\alpha(\alpha_1, ..., \alpha_k)$=e	Definición de función definida por el usuario; debe estar en un programa. Cada uno de α y α_1 a α_k es una sola letra o una sola letra seguida por «$» para argumento de cadena o resultado. Adopta la forma de **DEF FN** α()=e si no hay argumentos.
DELETE f	No opera sin Microdrive, etc.
DIM $\alpha(n_1, ..., n_k)$	Borra cualquier matriz con el nombre al y establece una matriz al de números con k dimensiones $n_1, ..., n_k$. Inicializa todos los valores a 0.
DIM α\$$(n_1, ..., n_k)$	Borra cualquier matriz, o cadena, con el nombre a$

	y establece una matriz de caracteres con k dimensiones n_1, ..., n_k. Inicializa todos los valores a " ". Ello puede considerarse como una matriz de cadenas de longitud fija n_k, con k-1 dimensiones n_1, ..., n_{k-1} Error 4 se produce si no hay espacio para almacenar la matriz. Una matriz no está definida hasta que se dimensione en una sentencia DIM.
DRAW x, y	**DRAW** x, y, 0
DRAW x, y, z	Dibuja una línea desde la posición de trazado actual desplazando x horizontalmente e y verticalmente con respecto a la misma, mientras gira a través de un ángulo z. Error B si se sale de la pantalla.
ERASE	No opera sin Microdrive, etc.
FLASH	Define si los caracteres serán parpadeantes o fijos. n=0 para situación estable y n=1 para parpadeante y n=8 para falta de cambio.
FOR α=x **TO** y	**FOR** al=x **TO** y **STEP** 1
FOR α=x **TO** y **STEP** z	Suprime cualquier variable simple al y establece una variable de control con valor x, límite y, paso z y dirección de iteración por bucle que se refiere a la sentencia después de la sentencia **FOR**. Comprueba si el valor inicial es mayor (si paso <0) o menor (si paso >0) que el límite y, de ser así salta a la sentencia **NEXT** α, dando error 1 si no hubiera ninguno. Ver **NEXT**. El error 4 se produce si no hay espacio para la variable de control.
FORMAT f	No opera sin Microdrive, etc.
GOSUB n	Introduce en una pila el número de línea de la sentencia **GOSUB**; luego, como **GO TO** n. El error 4 puede producirse si no hay espacio para la variable de control.
GO TO n	Salta a la línea n (o si no la hay, la primera línea después de ella).
IF x **THEN** s	Si x es verdadera (no cero), entonces, se ejecuta s. Observe que s comprende todas las sentencias hasta el final de la línea. No es admisible la forma '**IF** x **THEN**'
INK n	Establece el color de la tinta (primer plano) de los caracteres que posteriormente se van a imprimir. n está en el margen de 0 a 7 para un color, n=8 para transparente o 9 para contraste. Ver el apartado sobre la pantalla de televisión en el apéndice C(1). El error K se produce si n no está en el margenβ de 0 a 9.
INPUT ...	Los « ... » son una secuencia de elementos de **INPUT**,

separados como en una sentencia **PRINT** por comas, puntos y comas o apóstrofes. Un elemento de **INPUT** puede ser:

(i) Cualquier elemento de PRINT que no comience con una letra.
(ii) Un nombre de variable o
(iii) **LINE**, seguido de un nombre de variable de tipo cadena.

Los elementos de **PRINT** y los separadores en (i) se tratan exactamente como en **PRINT**, con la salvedad de que todo se imprime en la parte inferior de la pantalla.

Para (ii) el ordenador se detiene y espera la introducción de una expresión desde el teclado, cuyo valor se asigna a la variable. La entrada se refleja en la forma habitual y los errores de sintaxis se ponen de manifiesto por un símbolo ? intermitente. Para expresiones del tipo de cadena, la memoria intermedia de entrada se inicializa para contener dos comillas de cadena (que se pueden borrar si fuera necesario). Si el primer carácter de entrada es **STOP**, el programa se interrumpe con error H. (iii) es como (ii) con la salvedad de que la entrada se trata como un literal de cadena sin comillas y el mecanismo de **STOP** no tiene efecto; para detenerlo debe teclear ↓ en su lugar.

INVERSE n

Controla la inversión de caracteres que posteriormente se van a imprimir. Si n=0, los caracteres se imprimen en **video normal**, con el color de la tinta sobre el color del papel.
Si n = 1, los caracteres se imprimen en **video inverso**, esto es, color del papel sobre color de tinta. Ver el apartado sobre la pantalla de televisión en el apéndice c(1).

Error K si n no es 0 ni 1.

LET v=e

Asigna el valor de e a la variable v. **LET** no puede omitirse. Una variable simple no está definida hasta que es asignada a una sentencia **LET**, **READ** o **INPUT**. Si v es una variable de tipo cadena con subíndice o una variable de tipo cadena fragmentada o subcadena, entonces, la asignación es Procrusteana (de longitud fija): el valor de la cadena de e es truncado o rellenado con espacios a la derecha, para hacer que tenga la misma longitud que la variable v.

LIST

LIST 0

LLIST n

Lista el programa en la parte superior de la pantalla, comenzando en la primera línea cuyo número sea al menos n y convierte n en la línea en curso.

LLIST

LLIST 0

LLIST n

Como LIST, pero utilizando la impresora.

LOAD f	Carga programa y variables.
LOAD f **DATA** ()	Carga una matriz numérica.
LOAD f **DATA** $()	Carga matriz de caracteres $.
LOAD f **CODE** m,n	Carga, como máximo, n bytes, comenzando en la dirección m.
LOAD f **CODE** m	Carga bytes comenzando en la dirección m.
LOAD f **CODE** **LOAD** f **SCREEN$**	Carga bytes a partir de la dirección desde la que fueron salvados **LOAD** f **CODE** 16384,6912. Busca el fichero correcto en la cinta de cassette y lo carga, borrando las anteriores versiones en memoria. Ver capítulo 20.
LPRINT	Como **PRINT** pero utilizando la impresora.
MERGE f	Como LOAD f, pero no borra las variables y líneas de programa anterior, salvo para cargar las nuevas con el mismo nombre o número de línea.
MOVE f_1, f_2	No opera sin Microdrive, etc.
NEW	Comienza el sistema BASIC de nuevo, suprimiendo programa y variables y utilizando la memoria hasta e incluyendo el byte cuya dirección está en la variable del sistema **RAMBOT** y preserva las variables del sistema **UDG**, **P RAMT**, **RASP** y **PIP**.
NEXT α	(i) encuentra la variable de control α (ii) añade su incremento al valor (iii) si el incremento > 0 y el valor > límite; o si el incremento < 0 y el valor < límite, entonces, salta a la sentencia de control de bucle. Se produce error 2 si no hay ninguna variable α. Se produce el error 1 si no hay una variable, pero no es la variable de control α.
OPEN	No opera sin Microdrive, etc.
OUT m, n	Da salida al byte n en el 'port' m a nivel del procesador (Carga el par de registros bc con m, el registro a con n y realiza la instrucción de lenguaje ensamblador: out (c),a). 0 <= m <= 65535, -255 <= n <= 255; si no está entre los límites, se produce el error B.
OVER n	Controla la sobreimpresión para los caracteres que posteriormente se van a imprimir Si n =0, los nuevos caracteres borran los anteriores caracteres en esa posición. Si n = 1, entonces, los nuevos caracteres se mezclan con los antiguos para dar color de tinta cuando uno u otro (pero no ambos) tenían color de tinta, y color de papel si ambos fueran de color de papel o de color de tinta. Ver el apartado sobre la pantalla de televisión en el Apéndice c(1).

Se produce el error K si n no es 0 ni 1.

PAPER n

Como **INK**, pero controlando el color (fondo) del papel.

PAUSE n

Interrumpe el cálculo y congela el contenido de la pantalla durante n imágenes (a 50 imágenes por segundo o a 60 imágenes en Norteamérica) o hasta que se pulse una tecla.
0 <= n <= 65535; si estuviera fuera de estos límites, se produciría el error B. Si n=0, entonces la pausa no se temporiza, sino que dura hasta que se pulse una tecla.

PLOT C; m, n

Imprime una zona entintada (sujeta a **OVER** e **INVERSE**) en el pixel (m , n); mueve la posición del **PLOT**.
A no ser que los elementos de color c especifiquen otra cosa, el color de tinta en la posición de carácter que contiene el pixel se cambia al color de tinta permanente actual y las otras características (color de papel, parpadeo y brillo) se dejan inalteradas.
0 <= m <= 255, 0 <= n <= 175; si estuviera fuera de estos límites se produce el error B.

POKE m, n

Escriba el valor m en el byte de memoria de dirección m.

PRINT ...

Los ... es una secuencia de elementos de impresión **PRINT,** separados por comas, puntos y comas o apóstrofes y se escriben en pantalla.
Un punto y coma (;) entre dos elementos no tiene ningún efecto; se utiliza simplemente para separar los elementos. Una coma produce el carácter de control de coma y un apóstrofe produce el carácter **ENTER**. Al final de la sentencia **PRINT**, si no finaliza en un punto y coma, o coma, o apóstrofe, se imprime un carácter **ENTER**.

Un elemento de impresión **PRINT** puede ser:

(i) vacío; es decir, nada

(ii) una expresión numérica

Primero se imprime el signo menos, si el valor es negativo. Supongamos que sea x el módulo del valor a imprimir.
Si x <= 10^{-5} o x >= 1013 entonces, se imprime utilizando la notación científica. La parte de mantisa tiene hasta ocho dígitos (sin ningún cero a la derecha) y la coma decimal (ausente, si sólo hay un dígito) aparece después de la primera cifra. La parte del exponente es E, seguido por + o - y, a continuación, uno o dos dígitos.

En otro caso, x se imprime en notación decimal ordinaria con hasta ocho dígitos significativos y sin ceros a la derecha después de la coma decimal. Un punto (coma) decimal justo al principio

PRINT ...

siempre va precedido por un cero, así, por ejemplo, 3 y 0.3 se imprimen del mismo modo. 0 se imprime como un dígito úncio 0.

(iii) una expresión de cadena.

Los comandos («tokens») en la cadena están ex-expandidos, posiblemente con un espacio antes o después.

Los caracteres de control tienen su efecto de control.

Los caracteres no identificables se imprimen como **?**.

(iv) **AT** m,n

Da salida a un carácter de control **AT** seguido por un byte para m (el número de línea) y por un byte para n (el número de columna).

(v) **TAB** n.

Da salida a un carácter de control **TAB** seguido por dos bytes para n (el byte menos significativo primero), el de parada **TAB**.

(vi) Un elemento de color, que adopta la forma de una sentencia de **PAPER**, **INK**, **FLASH**, **BRIGHT, INVERSE** u **OVER**.

RANDOMIZE

RANDOMIZE 0

RANDOMIZE n

Establece la variable del sistema (llamada **SEED**) utilizada para generar el siguiente valor de **RND**. Si n <> 0, a **SEED** se le da el valor n; si n=0, entonces, se le da el valor de otra variable del sistema (llamada FRAMES-imágenes) que cuenta las imágenes hasta entonces presentadas por pantalla y por ello debe ser bastante aleatoria.

RANDOMIZE aparece como **RAND** en el teclado.

El error B se produce si n no está en el margen desde 0 a 65535.

READ v_1,v, ..., v_k

Asignada a las variables que utilizan las expresiones sucesivas en la lista de DATA.

Se produce el error C si una expresión es del tipo erróneo.

Se produce el error E si hay variables sin leer cuando se haya agotado la lista de **DATA**.

REM ...

Sin efecto. Los "...” pueden ser cualquier secuencia de caracteres, salvo **ENTER**. Puede incluir :, por lo que no son posibles sentencias después de la sentencia **REM** en la misma línea.

RESTORE

RESTORE 0

RESTORE n

Restaura el puntero de **DATA** a la primera sentencia **DATA** en una línea cuyo número sea al menos n: la siguiente sentencia **READ** comenzará la lectura allí.

RETURN

Toma la referencia de una sentencia en la pila **GO SUB** y salta a la línea después de ella.

	El error 7 se produce cuando no hay ninguna referencia de sentencia en la pila. Hay algún error en su programa. El número de sentencias **GO SUB** no corresponde con el número de setencias **RETURN**.
RUN	**RUN 0**
RUN n	**CLEAR** y luego, **GO TO** n
SAVE f	Conserva el programa y variables.
SAVE f **LINE**	Conserva el programa y las variables, de modo que si se cargan hay un salto automático al principio del programa.
SAVE f **LINE** m	Conserva el programa y las variables de modo que si se cargan, hay un salto automático a la línea m.
SAVE f **DATA** $()	Conserva la matriz numérica.
SAVE f **CODE** m,n	Conserva la matriz de caracteres $
SAVE f **SCREEN$**	Conserva n bytes comenzando en la dirección m.
	SAVE f **CODE** 16384,6192.
	Conserva información en cassette, dándolo el nombre f. Se produce el error F si f está vacío o tiene una longitud once o más carácteres. Ver capítulo 20.
STOP	Interrumpe el programa con informe 9. **CONTINUE** empezará de nuevo con la siguiente sentencia.
VERIFY	Lo mismo que **LOAD** con la salvedad de que los datos no se cargan en la **RAM,** sino que se comparan con las ya existentes allí. Se produce un error R si una de las comparaciones indica bytes diferentes.

Programas ejemplo

En este apéndice se incluyen algunos programas ejemplo para poner de manifiesto las capacidades funcionales del ZX Spectrum.

El primero de estos programs requiere la introducción de una fecha y proporciona el día de la semana que corresponde a esta fecha.

```
 10 REM conversión de fecha en día
 20 DIM d$(7,9): REM días de la semana
 30 FOR n=1 TO 7: READ d$(n): NEXT n
 40 DIM m(12): REM duraciones de los meses
 50 FOR n=1 TO 12: READ m(n): NEXT n
100 REM Introducción fecha
110 INPUT "día?";día
120 INPUT "mes?";mes
130 INPUT "año(siglo XX solamente)? ";año
140 IF año <1901 THEN PRINT "el siglo XX comienza en 1901":
    GO TO 100
150 IF año >2000 THEN PRINT "el siglo XX finaliza en el 2000":
    GO TO 100
160 IF mes <1 THEN GO TO 210
170 IF mes >12 THEN GO TO 210
180 IF año/4-INT( año/4)=0 THEN LET mes (2)=29: REM año
    bisiesto
190 IF día >m(mes) THEN PRINT "Este mes tiene solamente";
    m(mes);"días.": GO TO 500
200 IF día >0 THEN GO TO 300
210 PRINT "Imposible. Deme una fecha real".
220 GO TO 500
300 REM conversión fecha a número de días desde comienzo de siglo
310 LET a=año-1901
320 LET l=365*a+INT (a/4): REM número de días a comienzo de año
330 FOR n=1 TO mes-1: REM añadir a meses anteriores
340 LET l=l+m(n):NEXT n
350 LET l=l+día
400 REM conversión a día de la semana
410 LET l=l-7* INT (l/7)+1
420 PRINT día;"/";mes;"/";año
430 FOR n = 9 TO 3 STEP -1: REM quitar espacios a la derecha
440 IF d$(l,n) < > " THEN GO TO 460
450 NEXT n
460 LET e$=d$(l, TO n)
470 PRINT "era"; e$
500 LET m(2)=28: REM restaurar febrero
510 INPUT "desea conocer otra fecha?";a$
520 IF a$="n" THEN GO TO 540
```

```
 530 IF a$<>"N" THEN GO TO 100
1000 REM días de la semana
1010 DATA "Lunes", "Martes", "Miércoles"
1020 DATA "Jueves", "Viernes", "Sábado", "Domingo"
1100 REM duraciones de los meses
1110 DATA 31, 28, 31, 30, 31, 30
1120 DATA 31, 31, 30, 31, 30, 31
```

El siguiente programa trata de yardas, pies y pulgadas.

```
  10 INPUT "yardas?",yd,"pies?",ft,"pulgadas?",in
  40 GO SUB 2000: REM imprimirlos valores
  50 PRINT "igual a:";PRINT
  70 GO SUB 1000: REM el ajuste
  80 GO SUB 2000: REM imprimirlos valores ajustados
  90 PRINT
 100 GO TO 10
1000 REM subrutina para ajustar yd, ft, in a la forma normal
     para yardas, pies y pulgadas
1010 LET in=36*yd+12*ft+in: REM ahora todo está en pulgadas
1030 LET s=SGN in: LET in=ABS in: REM trabajamos con in
     positivo, reteniendo su signo en s
1060 LET ft=INT (in/12): LET in= (in-12*ft)*s: REM ahora in es correcto
1080 LET yd=INT (ft3)*s: LET ft=ft*s-3*yd: RETURN
2000 REM subrutina para imprimir yd, ft, in
2010 PRINT "yardas:";yd,,"pies:";ft,,"pulgadas:";in:PRINT: RETURN
```

A continuación se da un programa para tirar monedas para el 1 Ching. (Lamentablemente, produce las configuraciones al revés, pero podría no preocuparse por ello. En el capítulo 17 aprenderá sobre AT, que le permite imprimir desde abajo arriba).

```
   5 RANDOMIZE 10 FOR m=1 TO 6: REM para 6 tiradas
  20 LET c=0: REM inicializar total monedas a 0
  30 FOR n=1 TO 3: REM para 3 monedas
  40 LET c=c+2+INT (2*RND)
  50 NEXT n
  60 PRINT "   " ;
  70 FOR n=1 TO 2: REM para el hexagrama tirado, 2.° para los
     cambios
  80 PRINT "---";
  90 IF c=7 THEN PRINT "-" ;
 100 IF c=8 THEN PRINT " ";
 110 IF c-=6 THEN PRINT "X";: LET c=7
 120 IF c=9 THEN PRINT "0";: LET c=8
 130 PRINT "---  ";
 140 NEXT n
 150 PRINT
```

```
160 INPUT a$
170 NEXT m: NEW
```

Para utilizar este programa, introdúzcalo por el teclado y proceda a su ejecución y luego, pulse **ENTER** cinco veces para obtener los dos hexagramas. Observe estos últimos en un ejemplar del **Libro Chino de Cambios**. El texto describirá una situación y la secuencia de acciones adecuadas, usted debe reflexionar a fondo para descubrir el paralelismo entre ello y su propia vida. Pulse **ENTER** una sexta vez y el programa se borrará a sí mismo (ello le disuadirá a utilizarlo de forma frívola).

Muchas personas encuentran que los textos son siempre más aptos que lo que era de prever; este puede, o no puede, ser el caso de su ZX Spectrum. En términos generales, los ordenadores son criaturas bastante infieles.

Veamos, a continuación, un programa para jugar a los «pangolines». Piense un animal, y el ordenador intentará adivinar cuál es haciéndole preguntas que pueden contestarse «sí» o «no». Si el ordenador no tuvo noticia de su animal antes, le pide que le proporcione alguna pregunta que pueda utilizar en otra ocasión, para poder averiguarlo si alguien le plantease en el futuro la adivinación de su «nuevo» animal.

```
  5 REM pangolines
 10 LET nq=100: REM número de preguntas y animales
 15 DIM q$(nq,50): DIM a(nq,2): DIM r$(1)
 20 LET qf=8
 30 FOR n=1 TO qf/2-1
 40 READ q$(n): READ a(n,1): READ a(n,2)
 50 NEXT n
 60 FOR n=n TO qf-1
 70 READ q$(n): NEXT n

100 REM comienzo del juego
110 PRINT"Piense en un animal.","Pulse cualquier tecla para continuar."
120 PAUSE 0
130 LET c=1 : REM comienzo con la 1.ª pregunta
140 IF a(c,1)=0 THEN GO TO 300
150 LET p$=q$(c): GO SUB 910
160 PRINT "?": GO SUB 1000
170 LET in=1: IF r$="S"THEN GO TO 210
180 IF r$="S"THEN GO TO 210
190 LET in=2: IF r$="n" THEN GO TO 210
200 IF r$<>"N" THEN GO TO 150
210 LET c=a(c,in): GO TO 140

300 REM animal
310 PRINT "Pensaba en";
320 LET p$=q$(c): GO SUB 900: PRINT "?"
330 GO SUB 1000
340 IF r$="S" THEN GO TO 400
350 IF r$="S" THEN GO TO 400
360 IF r$="n" THEN GO TO 500
```

```
370 IF r$="N" THEN GO TO 500
380 PRINT "Respóndame adecuadamente cuando", "le esté hablando."
     GO TO 300
400 REM adivinado
410 PRINT "He pensado mucho.": GO TO 800

500 REM nuevo animal
510 IF qf > nq-1 THEN PRINT "Estoy seguro de que su animal es",
"muy interesante, pero no tengo", "espacio para ello ahora mismo."
     GO TO 800
520 LET q$(qf) =q$(c): REM cambio animal antiguo
530 PRINT "Qué es, entonces?":
540 PRINT Dígame algo que distinga entre

550 LET p$=q$(qf): GO SUB 900: PRINT " y "
560 LET p$=q$(qf+1): GO SUB 900: PRINT ""
570 INPUT s$: LET l=LEN s$
580 IF s$(l)="?"" THEN LET l=l-1
590 LET q$(c) =s$(TO l): REM insertar pregunta
600 PRINT "Entonces respuesta es"
610 LET p$=q$(qf+1): GO SUB 900: PRINT "?"
620 GO SUB 1000
630 LET in=1 : LET io=2: REM respuestas para animales nuevos
     y viejos "
640 IF r$="s" THEN GO TO 700
650 IF r$="s" THEN GO TO 700
660 LET in=2: LET io=1
670 IF r$="n" THEN GO TO 700
680 IF r$="N" THEN GO TO 700
690 PRINT "No es correcto ": GO TO 600
700 REM actualización respuestas
710 LET a(c,in)=qf+1: LET a(c,io)=qf
720 LET qf=qf+2: REM siguiente espacio de animal libre
730 PRINT "Eso me confundió".

800 REM de nuevo?
810 PRINT "Quiere otra oportunidad?": GO SUB 1000
820 IF r$="s" THEN GO TO 100
830 IF r$="s" THEN GO TO 100
840 STOP

900 REM PRINTimprimir sin espacios a la derecha
905 PRINT " ";
910 FOR n=50 TO 1 STEP -1
920 IF p$(n)<> " " THEN GO TO 940
```

```
 930 NEXT n
 940 PRINT p$(TO n);: RETURN

1000 REM " obtener respuesta
1010 INPUT r$: IF r$="" THEN RETURN
1020 LET r$=r$(1): RETURN

2000 REM animales primitivos
2010 DATA "'Vive en el mar" ,4,2
2020 DATA ""Es escamoso" ,3,5
2030 DATA "Come hormigas",6,7
2040 DATA "una ballena", "un tiburón blanco", "un pangolín",
     "una hormiga".
```

El siguiente programa es para dibujar una bandera inglesa («Union Jack»).

```
   5 REM bandera inglesa
  10 LET r=2: LET w=7: LET b=1
  20 BORDER 0: PAPER b: INK w: CLS
  30 REM negro en el fondo de la pantalla
  40 INVERSE 1
  50 FOR n=40 TO 0 STEP -8
  60 PLOT PAPER 0;7,n: DRAW PAPER 0;241,0
  70 NEXT n: INVERSE 0
 100 REM dibujar en partes blancas
 105 REM St. George (cruz)
 110 FOR n=0 TO 7
 120 PLOT 104+n,175: DRAW 0,-35
 130 PLOT 151-n,175: DRAW 0,-35
 140 PLOT 151-n,48: DRAW 0,35
 150 PLOT 104+n,48: DRAW 0,35
 160 NEXT n
 200 FOR n=0 TO 11
 210 PLOT 0,139-n: DRAW 111,0
 220 PLOT 255,139-n: DRAW -111,0
 230 PLOT 255,84+n: DRAW -111,0
 240 PLOT 0,84+n: DRAW 111,0
 250 NEXT n
 300 REM St. Andrew
 310 FOR n=0 TO 35
 320 PLOT 1+2*n,175-n: DRAW 32,0
 330 PLOT 224-2*n,175-n: DRAW 16,0
 340 PLOT 254-2*n,48+n: DRAW -32,0
 350 PLOT 17+2*n,48+n: DRAW 16,0
 360 NEXT n
 370 FOR n=0 TO 19
```

```
380 PLOT 185+2*n,140+n: DRAW 32,0
390 PLOT 200+2*n,83-n: DRAW 16,0
400 PLOT 39-2*n,83-n: DRAW 32,0
410 PLOT 54-2*n,140+n: DRAW -16,0
420 NEXT n
425 REM rellenar con bits adicionales
430 FOR n=0 TO 15
440 PLOT 255,160+n: DRAW 2*n-30,0
450 PLOT 0,63-n: DRAW 31-2*n,0
460 NEXT n
470 FOR n=0 TO 7
480 PLOT 0,160+n: DRAW 14-2*n,0
485 PLOT 255,63-n: DRAW 2*n-15,0
490 NEXT n
500 REM bandas rojas
510 INVERSE 1
520 REM St. George
530 FOR n=96 TO 120 STEP 8
540 PLOT PAPER r;7,n: DRAW PAPER r;241,0
550 NEXT n
560 FOR n=112 TO 136 STEP 8
570 PLOT PAPER r;n,168; DRAW PAPER r;0,-113
580 NEXT n
600 REM St. Patrick
610 PLOT PAPER r;170,140: DRAW PAPER r;70,35
620 PLOT PAPER r;179,140: DRAW PAPER r;70,35
630 PLOT PAPER r;199,83: DRAW PAPER r;56,-28
640 PLOT PAPER r;184,83: DRAW PAPER r;70,-35
650 PLOT PAPER r;86,83: DRAW PAPER r;-70,-35
660 PLOT PAPER r;72,83: DRAW PAPER r;-70,-35
670 PLOT PAPER r;56,140: DRAW PAPER r;-56,28
680 PLOT PAPER r;71,140: DRAW PAPER r;-70,35
690 INVERSE 0: PAPER 0: INK 7
```

Si no es británico, tiene la oportunidad de dibujar su propia bandera. Las tricolores son bastante fáciles, aunque algunos de los colores (por ejemplo, el naranja' de la bandera irlandesa) podrían presentar dificultades.

Veamos un programa para jugar al verdugo (también conocido como «juego del ahorcado»). Un jugador introduce una palabra y el otro ha de adivinarla.

```
 5 REM Juego del verdugo o del ahorcado
10 REM Preparar pantalla
20 INK 0: PAPER 7: CLS
30 LET x=240: GO SUB 1000: REM dibujar hombre
```

```
 40 PLOT 238,128: DRAW 4,0: REM boca
100 REM preparar palabra
110 INPUT w$: REM palabra a adivinar
120 LET l=LEN w$: LET v$=" "
130 FOR n=2 TO 1: LET v$=v$+" "
140 NEXT n: REM v$= palabra adivinada hasta ahora
150 LET c=0: LET d=0: REM contadores de errores y aciertos
160 FOR n=0 TO l-1
170 PRINT AT 20,n; "-";
180 NEXT n: REM escribir guiones "-" en lugar de letras
200 INPUT "Adivine una letra:";g$
210 IF g$="" THEN GO TO 200
220 LET g$=g$(1): REM 1.ª letra solamente
230 PRINT AT 0,c;g$
240 LET c=c+1: LET u$=v$
250 FOR n=1 TO l: REM actualizar palabra adivinada
260 IF w$(n)=g$ THEN LET v$(n)=g$
270 NEXT n
280 PRINT AT 19,0;v$
290 IF v$=w$ THEN GO TO 500: REM palabra adivinada
300 IF v$<>u$ THEN GO TO 200: REM la adivinación era correcta
400 REM dibujar siguiente parte de la horca
410 IF d=8 THEN GO TO 600: REM colgado
420 LET d=d+1
430 READ x0,y0,x,y
440 PLOT x0,y0: DRAW x,y
450 GO TO 200
500 REM hombre libre
510 OVER 1: REM borrar nombre
520 LET x=240: GO SUB 1000
530 PLOT 238,128: DRAW 4,0: REM boca
540 OVER 0: REM volver a dibujar el hombre
550 LET x=146: GO SUB 1000
560 PLOT 143,129: DRAW 6,0, PI/2: REM sonrisa
570 GO TO 800
600 REM colgar hombre
610 OVER 1: REM borrar suelo
620 PLOT 255,65: DRAW -48,0
630 DRAW 0,-48: REM abrir trampilla
640 PLOT 238,128: DRAW 4,0: REM borrar boca
650 REM Desmembrar
655 REM brazos
660 PLOT 255,117: DRAW -15,-15: DRAW -15,15
670 OVER 0
680 PLOT 236,81: DRAW 4,21: DRAW 4,-21
```

```
 690 OVER 1: REM piernas
 700 PLOT 255,66: DRAW -15,15: DRAW -15,-15
 710 OVER 0
 720 PLOT 236,60: DRAW 4,21: DRAW 4,-21
 730 PLOT 237,127: DRAW 6,0, -PI/2: REM Cara con ceño fruncido
 740 PRINT AT 19,0;w$
 800 INPUT "de nuevo?";a$
 810 IF a$="" THEN GO TO 850
 820 LET a$=a$(1)
 830 IF a$="n" THEN STOP
 840 IF a$(1)="N" THEN STOP
 850 RESTORE : GO TO 5
1000 REM dibujar hombre en columna x
1010 REM cabeza
1020 CIRCLE x,132,8
1030 PLOT x+4,134: PLOT x-4,134: PLOT x,131
1040 REM cuerpo
1050 PLOT x,123: DRAW 0,-20
1055 PLOT x,101: DRAW 0,-19
1060 REM piernas
1070 PLOT x-15,66: DRAW 15,15: DRAW 15,-15
1080 REM brazos
1090 PLOT x-15,117: DRAW 15,-15: DRAW 15,15
1100 RETURN
2000 DATA 120,65,135,0,184,65,0,91
2010 DATA 168,65,16,16,184,81,16,-16
2020 DATA 184,156,68,0,184,140,16,16
2030 DATA 204,156,-20,-20,240,156,0,-16
```

Binario y hexadecimal

En este apéndice se describe cómo cuentan los ordenadores, con el empleo del sistema binario.

La mayoría de los idiomas europeos utilizan para contar una configuración de decenas más o menos regular; en español, por ejemplo, aunque comienza un poco erráticamente, pronto se estabiliza en grupos regulares:

veinte, veintiuno, veintidós ... veintinueve
treinta, treinta y uno, treinta y dos ... treinta y nueve
cuarenta, cuarenta y uno, cuarenta y dos ... cuarenta y nueve

y así sucesivamente. Ello se hace incluso más sistemático con lo números arábigos que utilizamos en nuestros sistemas de numeración. Sin embargo, la única razón para emplear el número diez es que «casualmente» tenemos diez dedos.

En lugar de emplear el sistema **decimal**, con el número diez como base, los ordenadores utilizan una forma de binario denominada **hexadecimal** (o hex. en abreviatura), con base 16. Como sólo hay diez dígitos disponibles en nuestro sistema de numeración, neceistamos seis dígitos suplementarios para realizar el contaje. Por ello utilizamos A, B, C, D, E y F. ¿Y qué viene después de F? Lo mismo que nosotros, con diez dedos, escribimos 10 para diez, los ordenadores escriben 10 para dieciséis. Su sistema de numeración comienza:

Hex	Número cardinal
0	cero
1	uno
2	dos
.	.
.	.
9	nueve

lo mismo que en el sistema decimal pero, a partir de nueve, el sistema hexadecimal del ordenador es:

A	diez
B	once
C	doce
D	trece
E	catorce
F	quince
10	dieciséis
11	diecisiete
.	.
.	.
19	veinticinco
1A	veintiséis
1B	veintisiete
1C	veintiocho
.	.
.	.
1F	treinta y uno
20	treinta y dos

21	treinta y tres
.	.
.	.
9E	ciento cincuenta y ocho
9F	ciento cincuenta y nueve
A0	ciento sesenta
A1	ciento sesenta y uno
.	.
.	.
F4	ciento ochenta
.	.
.	.
FE	doscientos cincuenta y cuatro
FF	doscientos cincuenta y cinco
100	doscientos cincuenta y seis

Si está utilizando la notación hexadecimal y desea simplificar la indicación del sistema de numeración, puede escribir 'h' al final del número y decir «hex». Por ejemplo, para 158, escribir '9Eh' y diremos «nueve E hex».

Se estará preguntando qué tiene que ver todo ello con los ordenadores. De hecho, los ordenadores siempre se comportaron como si sólo tuvieran dos dígitos, representados por una baja tensión o desactivación (0) y una alta tensión o activación (1). Estos dos estados configuran el sistema **binario** y los dos dígitos binarios se llaman bits; por lo que un bit es 0 ó 1.

En los diversos sistemas, el contaje se iniciará como se indica a continuación:

Cardinal	**Decimal**	**Hexadecimal**	**Binario**
cero	0		101 ó 0101
uno	1	1	110 ó 0110
dos	2	2	111 ó 0111
tres	3	3	0 ó 0000
cuatro	4	4	1 ó 0001
cinco	5	5	10 ó 0010
seis	6	6	11 ó 0010
siete	7	7	100 ó 0011
ocho	8	8	1000
nueve	9	9	
diez	10	A	1010
once	11	B	1011
doce	12	C	1100
trece	13	D	1101
catorce	14	E	1110
quince	15	F	1111
dieciséis	16	10	10000

El punto importante es que dieciséis es igual a dos elevado a la cuartya potencia y ello facilita la conversión entre los sistemas hexadecimal y binario.

Para convertir hexadecimal en binario, hay que sustituir cada dígito hexadecimal por los cuatro bits correspondientes en binario, con el empleo de la tabla anterior.

Para convertir binario en hexadecimal, hay que dividir el número binario en grupos de cuatro bits, comenzando por la derecha y luego, sustituir cada grupo por el dígito hexadecimal correspondiente.

Por esta razón, aunque estrictamente hablando los ordenadores utilizan un sistema binario puro, los operadores suelen escribir los números almacenados en el interior de un ordenador con el empleo de la notación hexadecimal.

Los bits en el interior del ordenador suelen estar agrupados en conjuntos de ocho, o **bytes**. Un solo byte puede representar cualquier número desde cero a doscientos cincuenta y cinco (11111111 en binario o FF en hexadecimal) o, como alternativa, cualquier carácter incluido en el juego de caracteres del ZX Spectrum. Su valor puede escribirse con dos dígitos hexadecimales.

Dos bytes pueden agruparse para constituir lo que técnicamente se denomina una palabra. Una palabra puede escribirse utilizando 16 bits o 4 dígitos hexadecimales y representa a un número desde 0 a 216 - 1 = 65535 (en decimal).

Un byte siempre está constituido por ocho bits, pero las palabras varían en longitud de un ordenador a otro.

La notación **BIN** del capítulo 14 proporciona un medio de escribir números en binario en el ZX Spectrum: **«BIN 0»** representa a cero, **«BIN 1»** representa al uno, **«BIN 10»** representa al dos y así sucesivamente.

Sólo puede utilizar «0» y «1» para esta notación, por lo que el número debe ser un número entero no negativo; por ejemplo, no puede escribir **«BIN -11»** por -3, sino que ha de escribir **«-BIN 11»**. El número tampoco debe ser mayor que el decimal 65535 (esto es, no ha de tener más de 16 bits).

ATTR era realmente binario. Si convierte el resultado de ATTR en binario, puede escribirlo en ocho bits.

El primero es 1 para parpadeo y 0 para no parpadeo.

El segundo es 1 para brillo especial y 0 para brillo normal.

Los tres siguientes son el código para el color del papel, escrito en binario.

Los tres últimos son el código para el color de la tinta, escrito en binario.

En os códigos de colores también se utiliza el binario: cada código escrito en binario puede escribirse con tres bits, el primero para verde, el segundo para rojo y el tercero para azul.

Estos colores tienen un solo bit 1 en una de las tres posiciones. Sus códigos son 100, 010 y 001 en binario, o cuatro, dos y uno en decimal.

El negro no tiene nada visible, por lo que todos los bits son 0 (desactivación). Por consiguiente, el código para el negro es 000 en binario o cero.

Los demás colores son mezclas de los anteriores, por lo que sus códigos tienen dos o más bits 1.

Indice

Este índice incluye las teclas del teclado y cómo obtenerlas (el modo K, L, E, C o G y qué tecla de desplazamiento, cuando sea adecuada).

Habitualmente, una entrada es objeto de referencia una sola vez por capítulo, por lo que al haber encontrado una referencia, ojee el resto del capítulo incluyendo los ejercicios.

I

J

K

L

M

N

O

R

S

X

Y

Z



INVESTRONICA
Tomás Bretón, 21
MADRID-7 - SPAIN
Teléfs. 468 03 00 / 467 29 16
Télex: 23399IYCO E